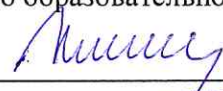


Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности



С.Т. Князев

« 7 »



2023 г.



## Алгоритмы и анализ сложности

Учебно-методические материалы по направлению подготовки  
**09.03.03 Прикладная информатика**  
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

## РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент департамента информационных технологий и автоматике



---

Трофимов С.П

Доцент департамента информационных технологий и автоматике



---

Трофимова О.Г.

## СОДЕРЖАНИЕ

Лекционный материал и лабораторные задания.....	4
Контрольная работа по дисциплине «Алгоритмы и анализ сложности».....	42
Перечень теоретических вопросов и заданий для итогового экзамена по дисциплине «Алгоритмы и анализ сложности» .....	43
Приложение 1. Пример отчета по лабораторной работе .....	48

**«Алгоритмы и анализ сложности»**  
**Лекционный материал и лабораторные задания**  
Трофимов С.П., Трофимова О.Г.

**ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ**

1. Понятие алгоритма (Алгоритм (лат. algorithmi — от имени среднеазиатского математика Аль-Хорезми) — конечная совокупность точно заданных правил решения некоторого класса задач или набор инструкций, описывающих порядок действий исполнителя для решения определённой задачи).

Классификация алгоритмов. Характеристики алгоритмов: сложность, порядок точности, сходимость, скорость сходимости, критерии останова. Библиотеки «плохих» задач.

Практическое вычисление сложности алгоритма. *Анализ бесконечно больших и бесконечно малых величин (ББВ и БМВ)*. Порядок малости/роста и коэффициент пропорциональности величины. Использование МНК для нахождения параметров функции сложности алгоритма.

Тестирование и отладка алгоритма. Тестирование алгоритма и демонстрация алгоритма – разный исходный код! Требования к тестированию. Процентные правила хорошего стиля программирования. Способы записи алгоритма. Способы обработки ошибок: вывод сообщений на консоль, запись в log-файл, иерархия исключений.

Характеристики машинных типов данных. Размер типа в байтах. Способ представления целых знаковых типов. Мантисса и порядок вещественных типов. Диапазон типов. Машинный ноль, машинный эпсилон и машинная бесконечность. Проблемы встроенных типов: выход за диапазон, переполнение и исчезновение порядка.

**1.1. Бесконечно малая величина (БМВ) и бесконечно большая величина (ББВ)**

**Определение 1.** Функция  $f(x)$  называется бесконечно малой величиной, если

$$\lim_{x \rightarrow 0} f(x) = 0 \quad (1)$$

Примеры БМВ:  $x, \sqrt{x}, x^2, x^3, x^4, \sin(x), \cos(x) - 1, \operatorname{tg}(x)$ .

В стандартных координатах графики БМВ в окрестности нуля практически неотличимы друг от друга, что не дает возможности провести визуальный анализ функций.

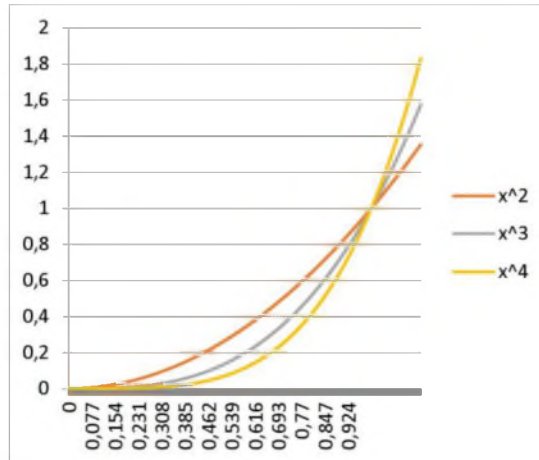


Рис.1 График БМВ в стандартных координатах

**Определение 2.** Порядком малости БМВ  $f(x)$  называется минимальное положительное число  $p$ , для которого предел

$$b = \lim_{x \rightarrow 0} \frac{f(x)}{x^p}$$

существует и отличен от нуля.

Примеры определения порядков малости:

1) БМВ  $y = x^2$  имеет второй порядок малости, так как

$$\lim_{x \rightarrow 0} \frac{f(x)}{x^2} = \lim_{x \rightarrow 0} \frac{x^2}{x^2} = 1 \neq 0.$$

2) БМВ  $y = \sqrt{x}$  имеет порядок малости  $\frac{1}{2}$ , так как  $\lim_{x \rightarrow 0} \frac{\sqrt{x}}{x^{1/2}} = 1 \neq 0.$

3) БМВ  $y = x$  имеет порядок малости 1, так как  $\lim_{x \rightarrow 0} \frac{x}{x} = 1 \neq 0.$

**Определение 3.** Функция  $F(x)$  – называется бесконечно большой величиной (ББВ), если  $\lim_{x \rightarrow \infty} F(x) = \infty.$

Между БМВ  $f(x)$  и ББВ  $F(x)$  существует следующая взаимосвязь:

$$f(x) = \frac{1}{F(1/x)}$$

является БМВ,

$$F(x) = \frac{1}{f(1/x)}$$

является ББВ.

БМВ могут появляться в различных случаях:

- 1) Задание с помощью некоторой аналитической формулы.
- 2) Для любой непрерывной функции  $g(x)$  и любой точки  $x_0$  величина  $f(x) = g(x+x_0) - g(x_0)$  является БМВ.
- 3) Погрешность численных методов.

### 1.2. Логарифмическая шкала.

Применение естественной шкалы для изображения графиков БМВ не позволяет рассмотреть поведение в малой окрестности нуля. Объяснение этому очевидному факту служит формула (1). Будем использовать для построения графика БМВ логарифмическую шкалу для обеих осей координат. Для этого прологарифмируем выражение  $y=f(x)$  по основанию 10. Получим

$$\lg y = \lg f(x). \quad (2)$$

Допустим правую часть выражения (2) удалось выразить через величину  $\lg x$ :

$$\lg f(x) = g(\lg x), \quad (3)$$

где  $g(x)$  некоторая аналитическая функция. Обозначим  $\bar{x} = \lg x$ ,  $\bar{y} = \lg y$ , тогда уравнение (3) принимает вид

$$\bar{y} = g(\bar{x}). \quad (4)$$

График функции (4) будем называть  $\lg$ - $\lg$  графиком исходной функции  $f(x)$

*Пример 1.* На рис. 2 изображен  $\lg$ - $\lg$  график функции  $y = x^{0,5}$ . Этот график совпадает со своей асимптотой, имеющей уравнение  $\bar{y} = 0,5 \cdot \bar{x}$ .

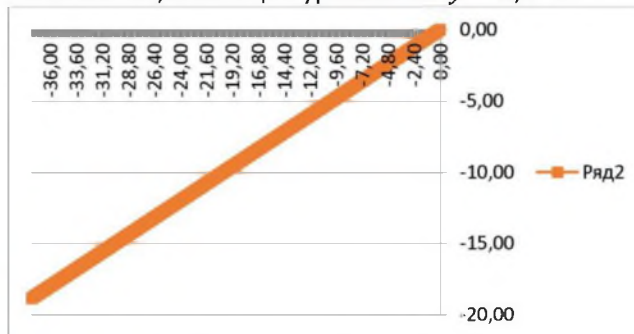


Рис.2.  $\lg$ - $\lg$  график БМВ  $y=x^{0,5}$

Таким образом, на горизонтальной оси координат  $\lg$ - $\lg$  графика вместо переменной  $x$ , близкой к нулю, будет изображаться логарифм этой величины  $\lg(x)$ , который при малых значениях  $0 < x < 1$  принимает значение из диапазона  $(-\infty; 0)$ . В результате  $\lg$ - $\lg$  графики двух различных БМВ можно наглядно сравнивать между собой.

### 1.3. Операции для $\lg$ - $\lg$ графиков.

Функцию  $g(x) = \lg f(x)$  будем называть  $\lg$  – изображением функции  $f(x)$  и представлять это соответствие в виде

$$g(\bar{x}) \leftrightarrow f(x).$$

Допустим, имеются две БМВ  $f_1(x)$  и  $f_2(x)$ . Построим их  $\lg$ -изображения:

$$\bar{y} = g_1(\bar{x}) = \lg f_1(x),$$

$$\bar{y} = g_2(\bar{x}) = \lg f_2(x).$$

Над  $\lg$ -изображениями можно осуществлять арифметические операции:

$$c g_1(\bar{x}) = c \lg f_1(x) = \lg f_1^c(x),$$

откуда вытекает

$$c g_1(\bar{x}) \leftrightarrow f_1^c(x)$$

Аналогично получаем

$$c + g_1(\bar{x}) \leftrightarrow 10^c \cdot f_1(x)$$

$$g_1(\bar{x}) + g_2(\bar{x}) \leftrightarrow f_1(x) \cdot f_2(x)$$

$$g_1(\bar{x}) \cdot g_2(\bar{x}) \leftrightarrow f_2(x)^{f_1(x)}$$

*Вывод:* Арифметические операции над логарифмическими образами функций БМВ соответствуют некоторым операциям над оригиналами исходных БМВ. Аналогичная ситуация имеет место с интегральным преобразованием Фурье.

**1.4. Асимптотический способ определения порядка малости БМВ.**

Определение 2 не дает способа определения порядка малости БМВ. Ниже описывается алгоритм нахождения не только порядка малости  $p$  БМВ  $f(x)$ , но и коэффициента  $b$  при соответствующем мономе  $b \cdot x^p$ . Таким образом, мы получаем наилучшее приближение

$$f(x) \approx b \cdot x^p.$$

Допустим, логарифмический образ некоторой БМВ  $f(x)$  задан функцией  $y = g(\bar{x})$  и имеет наклонную асимптоту

$$\bar{y} = a + k \cdot \bar{x} \quad (5)$$

где  $a$  и  $k$  - коэффициенты асимптоты. Известно (см., например, [1, с.119-122]), что эти коэффициенты определяются по формулам

$$k = \lim_{\bar{x} \rightarrow -\infty} \frac{g(\bar{x})}{\bar{x}} \quad (6)$$

$$a = \lim_{\bar{x} \rightarrow -\infty} [g(\bar{x}) - k\bar{x}] \quad (7)$$

**Теорема:** БМВ  $f(x)$  имеет порядок малости  $p$  тогда и только тогда, когда существует асимптота (5) для ее логарифмического образа (4). При этом

$$p = k, a = \lg(b). \quad (8)$$

**Вывод:** Таким образом, неизвестный заранее порядок малости БМВ можно найти по аналитическим формулам (6), (7), (8).

**Пример 2.** Из рис. 2 видно, что  $\lg\text{-}\lg$  график БМВ совпадает со своей асимптотой и равенство (8) выполняется.

Построим в Excel  $\lg\text{-}\lg$  график БМВ

Таблица. Макет листа Excel для построения  $\lg\text{-}\lg$  графика

x	f(x)	$\bar{x} = \lg x$	$g(\bar{x}) = \lg f(x)$
0.1	=f(0.1)	-1	=lgf(0.1)
0.01	=f(0.01)	-2	=lgf(0.01)
...	...	...	...
1E-19	=f(1E-19)	-19	=lgf(1E-19)

График строим по колонкам 3 и 4. После этого визуальным образом определяем асимптоту функции, находим две точки этой асимптоты и по ним определяем угловой коэффициент  $k$  асимптоты и свободный коэффициент  $a$ .

Существуют БМВ, для которых порядок малости существует, но не является конечным числом.

**Пример 3.** Рассмотрим БМВ  $y = x^{-\lg(x)}$ . После логарифмирования получим

$\lg(y) = -\lg(x) \cdot \lg(x)$ , то есть логарифмический образ  $\bar{y} = -\bar{x}^2$  является перевернутой параболой и не имеет наклонной асимптоты. Поэтому БМВ имеет бесконечный порядок малости, она стремится к нулю быстрее любой степенной функции. Примером ненулевой БМВ с нулевым порядком малости является функция  $y = 10^{-\sqrt{-\lg(x)}}$ .

**1.5. Порядок точности численных методов**

При анализе численных методов важную роль играет определение порядка точности этих методов. Например, порядок точности вычисления определенного интеграла методом трапеций равен 2. Это означает, что при уменьшении шага интегрирования в 10 раз погрешность метода уменьшится в 100 раз. Определение порядка точности метода является достаточно сложной математической задачей. Покажем, как можно это сделать с помощью нашего подхода.

Найдем точность двух методов вычисления производной  $f'(x_0)$ .

- 1) Дифференцирование с помощью правой формулы

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h}, \quad (10)$$

где шаг дифференцирования  $h$  теоретически является БМВ.

Функция полной погрешности формулы (10) имеет вид

$$\varphi(h) = \left| f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} \right| \quad (11)$$

Известно, что  $\varphi(h) \rightarrow 0$ , т.е.  $\varphi$  является бесконечно малой величиной.

Построим lg-lg график погрешности (11). Для нахождения асимптоты lg-lg графика найдем две ее точки. Из теории численных методов известно, что формула (10) имеет первый порядок малости, то есть  $\varphi(h) \approx C \cdot h$ . Отсюда  $\lg(\varphi(h)) \approx \lg(C \cdot h) = \lg(C) + \lg(h)$ , то есть угловой коэффициент к наклонной асимптоты должен быть равным  $k=1$ . Проверим эти рассуждения на примере.

*Пример 5.* Возьмем  $f(x)=x^2$ ,  $x_0=1$ . На рис. 4 представлен график погрешности по правой формуле (11).

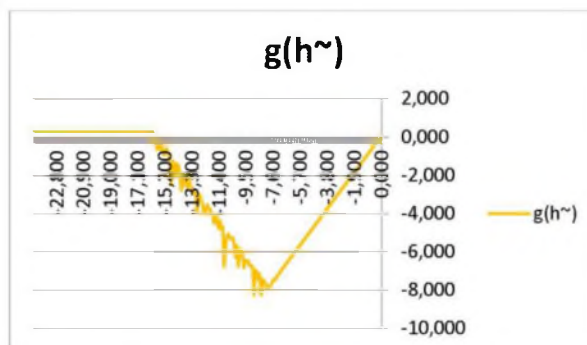


Рис. 4. График lg-lg погрешности правой формулы численного дифференцирования

Из рис.4 видно, что угловой коэффициент  $k=1$ .

- 2) Численное дифференцирование по центральной формуле.

$$f'(x_0) = \frac{f(x_0+h)-f(x_0-h)}{2h} \quad (12)$$

*Пример 6.* Возьмем для примера  $f(x) = x^3$ ,  $x_0 = 1$ . На рис. 5 представлен график погрешности по центральной формуле.



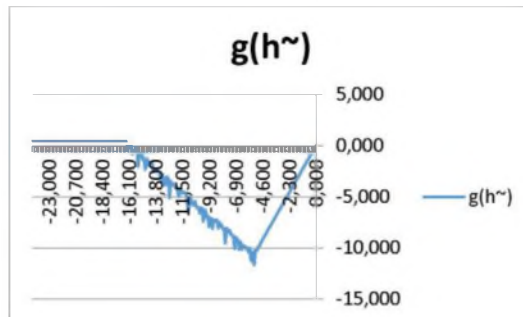


Рис. 5. График lg-lg погрешности центральной формулы численного дифференцирования

Из теории также известно, что формула (12) имеет второй порядок точности, то есть  $\lg \bar{x} = \lg(C) + 2 \cdot \lg(h)$ . Из рис.5 видно, что угловой коэффициент  $k=2$ .

При анализе бесконечно малых величин  $f(x)$  возникают дробные порядки малости  $\alpha$ , при которых

$$f(x) \approx C \cdot x^\alpha.$$

Определение числовых характеристик  $C$  и  $\alpha$  возможно с помощью методов дифференцирования дробных порядков.

## 2. Теоретико-числовые алгоритмы [Василенко].

Целочисленная арифметика произвольной точности. Дополнительный код для отрицательных чисел. Сложение и вычитание. Умножение, замена умножения несколькими суммами. Возведение в степень, преобразование показателя в двоичную систему, многократное возведение в квадрат. Деление [Василенко, стр. 254].

Модулярная арифметика. Операции модулярной арифметики произвольной точности.

Обратный элемент по модулю. Алгоритмы нахождения обратного элемента по модулю: алгоритм Эвклида, обобщенный алгоритм Эвклида [Василенко, 292], обобщенный бинарный алгоритм [Василенко, 300]

Решение модульного уравнения. Решение системы из двух модульных уравнений.

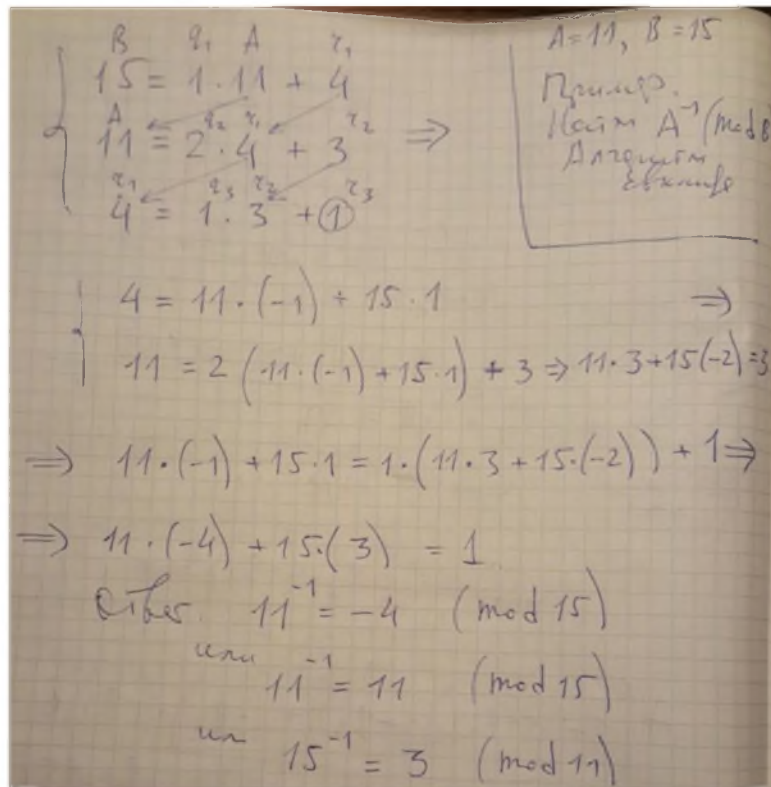
Китайская теорема об остатках. Алгоритм Гарнера решения китайской теоремы.

Алгоритм RSA-шифрования

### Обратный элемент по модулю

$A=11, B=15$ . Найти  $A^{-1} \pmod{B}$  с помощью алгоритма Эвклида

### Алгоритм решения уравнения $ax+by = 1$ .



1. Определим матрицу E:

$$E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

2. Вычислим  $r$  - остаток от деления числа  $a$  на  $b$ ,  $a=bq+r$ ,  $0 \leq r < b$ .

3. Если  $r=0$ , то второй столбец матрицы E даёт вектор  $(x, y)$  решений уравнения.

4. Если  $r \neq 0$ , то заменим матрицу E матрицей

$$E^* = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$$

5. Заменим пару чисел  $(a, b)$  на  $(b, r)$  и перейдем к шагу 2.

### Китайская теорема об остатках.

**Пример.** Решение системы двух модулярных уравнений

$$\begin{cases} X \equiv a_1 \pmod{m_1} \\ X \equiv a_2 \pmod{m_2} \end{cases}$$

Система двух  
линейных уравнений

$$\begin{cases} X = k_1 m_1 + a_1 \\ X = k_2 m_2 + a_2 \end{cases} \text{ Неполные } k_1, k_2.$$

$$k_1 m_1 + a_1 = k_2 m_2 + a_2$$

$$k_1 m_1 - k_2 m_2 = a_2 - a_1.$$

Тогда

Находим  $k_1^0, k_2^0$  :  
 $k_1^0 m_1 - k_2^0 m_2 = 1.$

$$k_1^0 (a_2 - a_1) \cdot m_1 + k_2^0 (a_2 - a_1) \cdot m_2 = a_2 - a_1$$

Находим  
 $k_1 = k_1^0 (a_2 - a_1)$

$$k_2 = k_2^0 (a_2 - a_1).$$

Ответ:  $X = k_1^0 (a_2 - a_1) \cdot m_1 + a_1$

Пример  $X \equiv 2 \pmod{3}$   
 $X \equiv 5 \pmod{7}$

1). Неполные уравнения  $k_1^0 \cdot 3 - k_2^0 \cdot 7 = 1$   
 $5 \cdot 3 - 2 \cdot 7 = 1$   
 $k_1^0 = 5, k_2^0 = 2$

2).  $X = 5 \cdot (5-2) \cdot 3 + 2 = 5 \cdot 3 \cdot 3 + 2 = 47$

$$47 \equiv 2 \pmod{3} \quad \checkmark$$

$$47 \equiv 5 \pmod{7} \quad \checkmark$$

$$(*) \begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \end{cases} \Rightarrow \text{Король } \tilde{x} \text{ 2.}$$

$\Rightarrow$  ~~∃~~ одн. реш.

$$x = \tilde{x} + k \cdot m_1 \cdot m_2$$

$$(*) \Rightarrow \begin{cases} x \equiv \tilde{x} \pmod{m_1 \cdot m_2} \\ x \equiv a_3 \pmod{m_3} \end{cases}$$

аурыс-д

Обобщенный Суньцзю арифметика стр 300

$$m_1^{-1}: \quad m_1 \cdot x \equiv 1 \pmod{m_2}, \quad m_1 < m_2$$

$$m_1 \cdot x - m_2 \cdot k = 1.$$

$$x = m_1^{-1} \pmod{m_2}$$

Обобщенная арифметика. Евануш стр 292.

$$\begin{cases} u_1 a + u_2 b = u_3 \\ v_1 a + v_2 b = v_3 \\ t_1 a + t_2 b = t_3 \end{cases}$$

Доказательство того, что Боб правильно восстановил оригинальный текст Алисы приводится в [Dreese]. Но если оно (доказательство) вам не сильно интересно, то можете его пропустить и просто поверить, что это так.

Пример.

Шаг 1.  $p=3, q=7$ . модуль  $n=p*q=21$ . Боб вычисляет  $fi = (p-1)*(q-1) = 12$ . открытая экспонента  $e=5$ . Боб пересылает Алисе открытый ключ  $\{5, 21\}$

Боб вычисляет  $d = e^{-1}(\text{mod } fi) = 5^{-1}(\text{mod } 12) = 17$ . Таких  $d$  много, берем любой. Боб получил закрытый ключ  $\{d, n\} = \{17, 21\}$

Шаг 2. Пусть  $P = 19$ .  $E = P^e (\text{mod } n) = 19^5(\text{mod } 21) = 10$ . Алиса пересылает Бобу открыто  $E = 10$

Шаг 3. Боб вычисляет оригинальный текст

$P = 10^d (\text{mod } n) = 10^{17}(\text{mod } 21)=19$ .

3. Универсальная машина Тьюринга. Машина Маркова. Сложность алгоритмов. Эквивалентные задачи. N-сложные, NP-сложные, P-сложные, NP-трудные, NP-полные классы алгоритмов и задач.

Примеры программ для решения простых задач на машине Тьюринга.

4. Рекурсия. Схема стека вызовов рекурсии. Однократные и многократные рекурсии. Использование ресурсов компьютера: время и память. Методы разделяй и властвуй. Сложность рекурсивных алгоритмов и рекуррентные соотношения. Функциональное уравнение рекурсии. Примеры рекурсивных алгоритмов: разложение числа на простые множители, вычисление определителя, распознавание формулы в строке.

5. Быстрые алгоритмы. Быстрое умножение длинных чисел. Быстрое произведение матриц. Быстрое дискретное преобразование Фурье.

6. Алгоритмы сортировки. Сортировка пузырьком. Быстрая сортировка. Алгоритм сортировки СортДерево. Красно-черная сортировка. Сложность алгоритмов.

7. Методы структурирования и извлечения данных для алгоритмов. Иерархический способ хранения данных с помощью xml-файлов. Структура xml-файлов. Виды узлов. Библиотека классов для работы с xml-файлами. Язык запросов к xml-файлам.

8. Динамические структуры данных. Структура динамического узла: информационные и служебные поля. Интерфейс динамической структуры данных. Шаблонные классы. Библиотека шаблонных классов STL. Итераторы.

9. Комбинаторные алгоритмы. Методы полного перебора: генерация перестановок, генерация подмножеств. Алгоритм Грея генерации подмножеств.

**9.1. Алгоритм Нарайаны** — нерекурсивный алгоритм, генерирующий по данной перестановке следующую за ней перестановку (в лексикографическом порядке). Придуман индийским математиком Пандитом Нарайаной в XIV веке.

Если алгоритм Нарайаны применить в цикле к исходной возрастающей последовательности из  $n$  элементов, то он сгенерирует все перестановки элементов в лексикографическом порядке.

Другой особенностью алгоритма является то, что необходимо запоминать только один элемент перестановки.

Алгоритм Нарайаны

Шаг 1: найти такой наибольший  $j$ , для которого  $a_j < a_{j+1}$ .

Шаг 2: увеличить  $a_j$ . Для этого надо найти наибольшее  $l > j$ , для которого  $a_l > a_j$ . Затем поменять местами  $a_j$  и  $a_l$ .

Шаг 3: записать последовательность  $a_{j+1}, \dots, a_n$  в обратном порядке.

Оценка сложности

В случае перестановки, где элементы перемешаны случайным образом, сложность алгоритма практически не зависит от количества элементов. В приведённых реализациях производится в среднем 3 сравнения элементов перестановки и 2.17 обменов.

Наилучшим является случай, когда предпоследний элемент перестановки больше последнего, тогда производится 2 сравнения и 1 обмен. Худшим является случай, когда элементы перестановки отсортированы по убыванию, и, если длина перестановки равна  $n$ , то производится  $n+1$  сравнений и  $n/2+1$  обменов.

В целом сложность алгоритма можно оценить как  $O(n)$  [Knuth].

**9.2. Алгоритм построения бинарного кода Грея.** Алгоритм генерирует все подмножества  $n$  – элементного множества, при чем каждое следующее подмножество получается из предыдущего путем добавления или удаления одного элемента [Федоряева].

10. **Динамическое программирование.** Принцип оптимальности Беллмана. Прямой и обратный ход в алгоритмах динамического программирования. Поиск наибольшей общей подстроки. Поиск минимального пути в графе.

*Принцип оптимальности Беллмана.* Подпуть максимального пути является максимальным.



## Поиск минимального пути в графе

Графа дорожная сеть с переходами и дорогами.

$x^0$  - начальная точка,

$x^n$  - финальная точка.

Требуется найти путь у  $x^0$  в  $x^n$  минимальной длины.

Определение. Окружность  $i$ -го радиуса,  $i = \overline{1, n}$ , называется множеством  $S_i$  точек, в которые можно прийти у  $x^0$  за  $i$  переходов

$$S_i = \{ x_j^i : j \in I_i \}.$$

При этом предполагается, что множества  $S_i$ ,  $i = \overline{1, n}$ , не пересекаются.

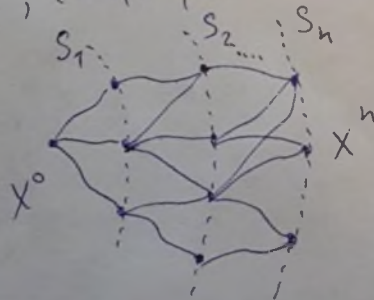


Рис. 1. Граф и его окружности

Минимальное расстояние от  $x^0$  до  $x^n$  обозначим  $g(x, y)$ .

Принцип Беллмана.

$$S(x^0, x^n) = \min_{k \in I_{n-1}} \{ S(x^0, x_k^{n-1}) + S(x_k^{n-1}, x^n) \} \quad (1)$$

Равенство (1) реализует "прямой ход" алгоритма динамического программирования (ДП). Для "обратного хода" используется равенство

$$S(x^0, x^n) = \min_{k \in I_1} \{ S(x^0, x_k^1) + S(x_k^1, x^n) \} \quad (2)$$

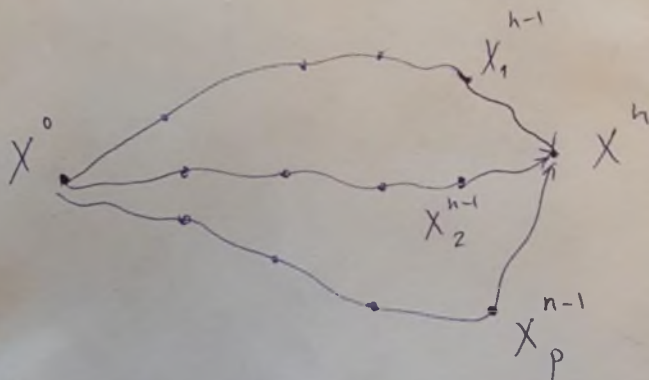


Рис. 2. Прямой ход ДП,  $p = |I_{n-1}|$ .



Поиск минимального пути  
в графе. Алгоритм Дейкстры

Задан взвешенный граф

$$G = \langle V, E, L \rangle,$$

где ребра  $u \in E$  соединяют вершины  $u \in V$  и имеют вес/длину  $l$ .

Начальная вершина  $A$ ,

конечная вершина  $B$ .

Ребра могут быть ориентированы.

Определение 1. Меткой вершины

$v \in V$  называется пара  $\langle l, v_{\text{пред}} \rangle$ ,

где  $l$  - длина некоторой пути  $u$

из  $A$  в  $v$ ,  $v_{\text{пред}}$  - предпоследняя вершина в этой пути.

Определение 2. Вершина с меткой

называется помеченной.

Определение 3. <sup>Помеченная,</sup> Вершина, в метке

которой длина пути  $l$  - минимальна,

называется просмотренной.

Определение 4. Начальная вершина  $A$

считается просмотренной. Множество

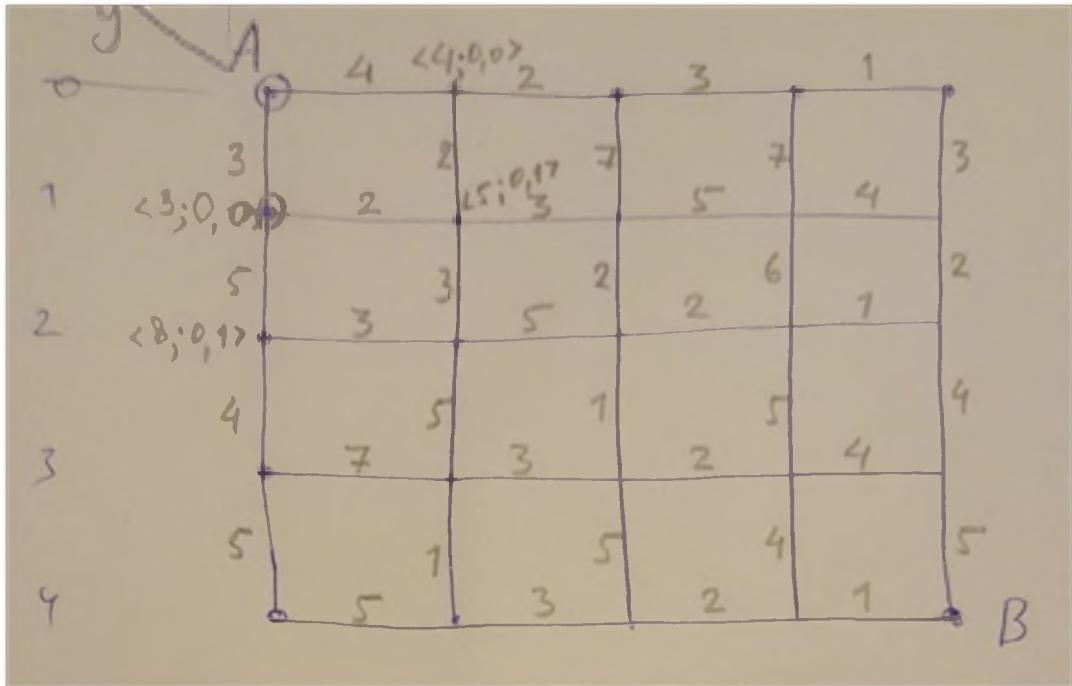
всех просмотренных вершин

обозначим  $\Pi$ .

Определение. Оболочка множества  $\Pi$  называется оболочкой минимальных расстояний  $1$  от  $\Pi$ . Обозначим оболочку  $\Theta(\Pi)$ .

### Алгоритм Дейкстры.

1.  $\Pi = \{A\}$
2. Находим оболочку  $\Theta(\Pi)$  и помечаем для каждой метки. Исполняем путь в одно ребро.
3. В оболочке  $\Theta(\Pi)$  находим вершину  $\tilde{v}$  с минимальной длиной  $l$  в метке.
4. Называем  $\tilde{v}$  просмотренной, т.е. включаем  $\tilde{v}$  в  $\Pi$ .
5. Если финальная вершина  $B \in \Pi$ , то длина мин. пути найдена и равна длине  $l$  метки  $B$ .  
Сам путь определяется с помощью  $\Theta$  опер. STOP.
6. Перейти к п. 2).



Задача. 0 разном монет. Имеется неограниченное количество монет достоинством 1, 2, 3, 5, 7 у.е. Найти минимальное по количеству монет разное суммы  $S=20$  у.е.

Решение "Прямой ход". 1) Составим таблицу "слева направо".

last \ $\Sigma$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1	1	2	2	2	3	2	-	2	3	3	3	4	3	4	5	4	4	4	4	5	4
2	-	1	2	2	2	2	-	3	2	3	3	3	4	3	4	3	4	4	4	4	5
3	-	0	1	2	2	2	-	2*	2	2	3	3	3	4	3	4	3	4	4	4	4
5	-	0	0	0	1*	2	-	2	3	2	3	2	3*	3	3	4	3	4	3	4	4
7	-	0	0	0	0	0	1	2	2	2	3	2	3	2	3	3	3	3	4	3	4*

Таблица 1. Минимальное количество монет в разном сумме  $\Sigma$ , last - последняя монета в разном.

- Находим миним. число в колонке  $S=20$ . Тогда последняя монета в разном 7. Превышая сумма  $S=20-7=13$
- Находим миним. число в колонке  $S=13$ . Последняя монета = 5. Превышая сумма  $S=13-5=8$
- $S=8$ . Последняя монета = 3.  $S=8-3=5$
- $S=5$ . Послед. монета = 5.  $S=5-5=0$ . **STOP**.

Ответ: Минимальный разном  $20=7+5+3+5$



"Обратный кеш."

1) Составляем таблицу "справа-налево"

first \ $\Sigma$	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	-	1*	2	-	2	3	2	3	2	3	3	3	4	3	4	3	4	4	4	5	4
2	-	-	1	-	2	2	3	2	3	2	3	3	3	4	3	4	3	4	4	4	5
3	-	-	-	1	2	2	2	3	2	3	2	3	3	3	4	3	4	3	4	4	4
5	-	-	-	-	-	1	2*	2	2	3	2	3	2	3	3	3	4	3	4	3	4
7	-	-	-	-	-	-	-	1	2	2	2	3	2	3*	2	3	3	3	4	3	4*

Табл 2. Обратный кеш. Минимальное количество монет в разменке. Минимальная сумма  $\Sigma$ , first - первая монета в разменке

- Находим миним. число в колонке  $\Sigma=0$ , first=7, Матричная сумма  $\Sigma=7$
- Находим миним. число в колонке  $\Sigma=7$ , first=7
- Находим миним. число в колонке  $\Sigma=14$
- Колонка  $\Sigma=14$ , first=5. Набрали  $\Sigma=19$
- Колонка  $\Sigma=19$ , first=1. Набрали  $\Sigma=20$ . STOP

След: Минимальный размен 20 = 7 + 7 + 5 + 1

## 11. Эволюционные алгоритмы. Генетический алгоритмы.

### Генетический алгоритм.

Хромосома = Особь состоит из генов.

Генотип хромосомы = последовательность значений генов.

Значение гена = аллель.

Фенотип хромосомы = предметная интерпретация генотипа.

Популяция = набор особей.

Родительский пул = множество с повторениями из популяции, участвующее в создании потомков.

Новая популяция = собранная из старой популяции и потомков.

Выбор Родительский пул (селекция):

- рулетка, использует величину приспособленности особи старой популяции для определения вероятности ее включения в пул;

- турнирный метод.

Отбор особей в новую популяцию:

- Конкурентный, берутся лучшие из Родителей + Потомки;

- Неконкурентный, берутся лучшие из Родителей + лучшие из Потомки.

Кроссигвер или скрещивание пары особей = обмен хвостами.

Выбор пары родителей для скрещивания: случайная пара.

Ген скрещивания – случайный ген.

Мутация – задается вероятность мутации для каждой особи.

Ген мутации выбирается случайно.

Бинарные гены. Генотип Особи = двоичное число. Фенотип особи – предметное значение особи. Для кодирования особей надо использовать код Грея.

Код Грея записывает двоичные числа в порядке, при котором соседние числа отличаются одним битом.

Результат алгоритма: улучшается среднее значения приспособленности популяции

Непрерывный генетический алгоритм.

Минимизация функции одной переменной.

Минимизация функции нескольких переменных.

12. Алгоритмы теории вероятностей. Операции со случайными величинами. Алгоритмы построения генераторов одномерных и многомерных случайных величин. Числовые характеристики одномерных и двумерных выборочных совокупностей.

13. Алгоритмы статистического анализа данных. Проверка статистических гипотез [Калугина]. Кластерный анализ.

14. Вычислительные алгоритмы. Численное дифференцирование и интегрирование. Алгоритмы решения дифференциальных уравнений. Алгоритмы нахождения корней многочленов, в т. ч. комплексных. Вычисление матричных характеристик. Решение уравнений.

15. Численные методы оптимизации. Оптимальные алгоритмы одномерной оптимизации. Алгоритм золотого сечения. Оптимизация по направлению для функции нескольких переменных. Метод наискорейшего спуска.

16. Геометрические алгоритмы. Математическая и геометрическая система координат. Иерархия классов геометрических примитивов. Реализация полиморфизма в стиле C++ с помощью механизма виртуальных функций. Алгоритм построения выпуклой оболочки точек. Алгоритм вычисления площади многоугольника. Принадлежность точки многоугольнику. Невидимые линии трехмерных многогранников.

17. Алгоритмы работы с нечеткими данными [Лавров, Леоненков]

Нечеткие множества. Нечеткие отношения. Нечеткие числа. Нечеткая логика. Нечеткий логический вывод. Этапы нечеткого логического вывода.

### Требования к отчету по лабораторной работе

1. Титульный лист:
    - шапка с названием учебного заведения;
    - название работы;
    - отчет по лабораторной работе дисциплины «Алгоритмы, структуры данных и анализ сложности»
    - Студент: ФИО, группа
    - Преподаватель: ФИО, должность, степень
    - Год
  2. Оглавление
  3. Задание
  4. Теоретическая часть
  5. Инструкция пользователя
  6. Инструкция программиста
  7. Тестирование
  8. Выводы
  9. Литература
  10. Приложение
- ✓ Задача должна быть решена в виде отдельной функции или метода класса.
  - ✓ Функция должна иметь тестирующие функции.
  - ✓ Исходные данные должны передаваться в функцию в виде структуры или отдельных параметров.
  - ✓ Исходные данные передаются программе в структурированном виде – в формате xml-файла.
  - ✓ Студент должен выполнить 4 лабораторные работы. Работы №№1,2 являются обязательными. Еще две работы выбираются студентом по согласованию с преподавателем.

### Лабораторная работа №1\*

#### Целочисленная арифметика произвольной точности и RSA-шифрование

1. Написать класс, который содержит целое число со знаком в виде массива однобайтовых элементов. Реализовать конструкторы, деструктор, перегрузить операции: аддитивные (+, -), мультипликативные (\*, /, %), сравнения (==, !=, <, >), взятие обратного по заданному модулю.
2. Написать функцию шифрования строки с помощью алгоритм RSA.
3. Зашифровать/расшифровать текстовый файл с помощью открытого RSA-ключа.

### Лабораторная работа №2\*

#### Алгоритмы сортировки строк

Реализовать один из алгоритмов сортировки строк:

1. Сортировка пузырьком
2. QSort
3. SortTree
4. Сортировка вставками
5. Сортировка слиянием
6. Пирамидальная сортировка HeapTree
7. Поразрядная сортировка
8. Сортировка с помощью красно-черного дерева

Выбор алгоритма выбирается по согласованию с преподавателем.

Для алгоритма определить сложность относительно наиболее характерной операции (сравнение, перестановка и др.). Вид функции сложности  $F(n)$  подобрать в соответствии с теорией. Например, для оптимальных алгоритмов  $F(n) = C * n * \log_2(n)$ . Найти также коэффициент пропорциональности  $C$ . Для аппроксимации можно использовать метод наименьших квадратов и сервис «Поиск решения».

План проведения эксперимента с алгоритмом называется *массовой задачей*. Представьте план в виде xml-файла. Например,

```
<experiments>
  <experiment name="Bubble">
    <nodes name="Arithmetic Progression" minElement=0 maxElement=20
startLength="1000" diff="100" maxLength="3000" repeat="15">
      <\nodes>
        <nodes name="Geometric Progression" minElement=0 maxElement=20
startLength="10" Znamen="1.5" maxLength="10000" repeat="15">
          <\nodes>
            <\experiment>
  .....
```

Длины массивов образуют арифметическую (или геометрическую) прогрессию начальным значением startLength, разностью diff (знаменатель Znamen) и

3. Использует операцию сравнения

#### 4. Сортировка вставками

Алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Используется бинарный поиск для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

Особенности алгоритма:

1. Сложность алгоритма  $O(n \cdot \log n)$ .
2. Используется дополнительная память  $n$  ячеек для хранения отсортированного массива.

#### 5. Сортировка слиянием

Алгоритм был изобретён Джоном фон Нейманом в 1945 году

Рекурсивная сортировка выполняется в три этапа:

1. Массив разбивается на два подмассива примерно одинакового размера;
2. Каждый подмассив сортируется отдельно;
3. Два упорядоченных подмассива половинного размера соединяются в один.

Основная идея слияния двух отсортированных массивов. Пусть мы имеем два уже отсортированных по возрастанию подмассива.

На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Особенности алгоритма:

1. Устойчивый - сохраняет взаимное расположение равных элементов.
2. Требуется дополнительной памяти по размеру исходного массива.

#### 6. Пирамидальная сортировка HeapTree

Пирамидальная сортировка (англ. Heapsort, «Сортировка кучей»).

Предложена Дж. Уильямсом в 1964 году.

Сортировка пирамидой использует бинарное сортирующее дерево.

Сортирующее дерево — это такое дерево, у которого выполнены условия:



1. Каждый лист имеет глубину либо  $d-1$ , либо  $d$ , где  $d$  — максимальная глубина дерева.
2. Значение в любой вершине не больше значения её двух потомков.

Удобная структура данных для сортирующего дерева — такой массив  $Array$ , что  $Array[0]$  — элемент в корне, а потомки элемента  $Array[i]$  являются  $Array[2i+1]$  и  $Array[2i+2]$ , где  $i = 0, \dots, n/2-1$ .

Алгоритм сортировки будет состоять из двух основных шагов:

1. Выстраиваем элементы массива в виде сортирующего дерева:  
 $Array[i] \leq Array[2i+1]$ ,  
 $Array[i] \leq Array[2i+2]$   
при  $i = n/2-1, \dots, 0$ .

Для этого сравниваем  $Array[i]$  с  $\min\{Array[2i+1], Array[2i+2]\}$  и при необходимости переставляем.

Этот шаг требует  $O(n \cdot \log(n))$  операций.

2. Будем удалять элементы из корня по одному за раз и перестраивать дерево. То есть на первом шаге обмениваем  $Array[0]$  и  $Array[n-1]$ , преобразовываем  $Array[0], Array[1], \dots, Array[n-2]$  в сортирующее дерево. Затем переставляем  $Array[0]$  и  $Array[n-2]$ , преобразовываем  $Array[0], Array[1], \dots, Array[n-3]$  в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда  $Array[0], Array[1], \dots, Array[n-1]$  — упорядоченная последовательность.

Этот шаг требует  $O(n \cdot \log n)$  операций.

Особенности алгоритма:

1. Работает в худшем, в среднем и в лучшем случае (то есть гарантированно) за  $\Theta(n \log n)$  операций при сортировке  $n$  элементов.
2. Количество применяемой служебной памяти не зависит от размера массива (то есть,  $O(1)$ ).
3. Из-за сложности алгоритма выигрыш получается только на больших  $n$
4. Неустойчив

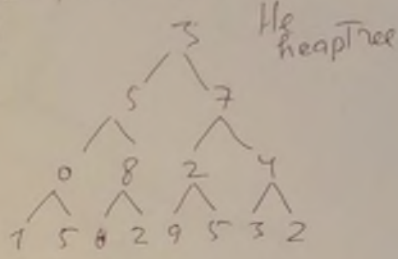
Пример.

Пример (HeapTree)

$A = \{ 3, 5, 7, 0, 8, 2, 4, 1, 5, 6, 2, 9, 5, 3, 2 \}$

$n = 15, n/2 = 7, i = n/2 - 1, \dots, 0 = 6, \dots, 0$

Q:  $A_i \leq \min \{ A_{2i+1}, A_{2i+2} \} ?$



I. Этап построения heapTree

1) Проверим  $i = 6$   
 $A_6 \leq \min \{ A_{13}, A_{14} \} ?$  нет.  
 (4) (3) (2)

$A = \{ 3, 5, 7, 0, 8, 2, 2, 1, 5, 6, 2, 9, 5, 3, 4 \}$

2)  $i = 5$ .  $A_5 \leq \min \{ A_{11}, A_{12} \} ?$  ✓

3)  $i = 4$ ,  $A_4 \leq \min \{ A_9, A_{10} \} ?$  нет.  
 (8) (6) (2)

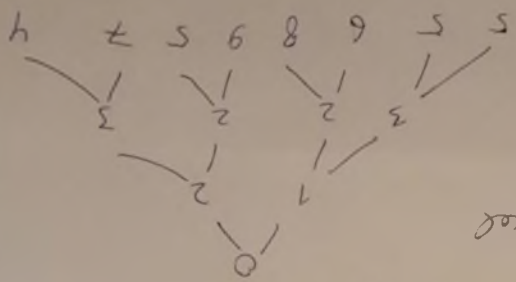
$A = \{ 3, 5, 7, 0, 2, 2, 2, 1, 5, 6, 8, 9, 5, 3, 4 \}$

4)  $i = 3$ .  $A_3 \leq \min \{ A_7, A_8 \} ?$  да

$A = \{ 3, 5, 7, 0, 2, 2, 2, 1, 5, 6, 8, 9, 5, 3, 4 \}$

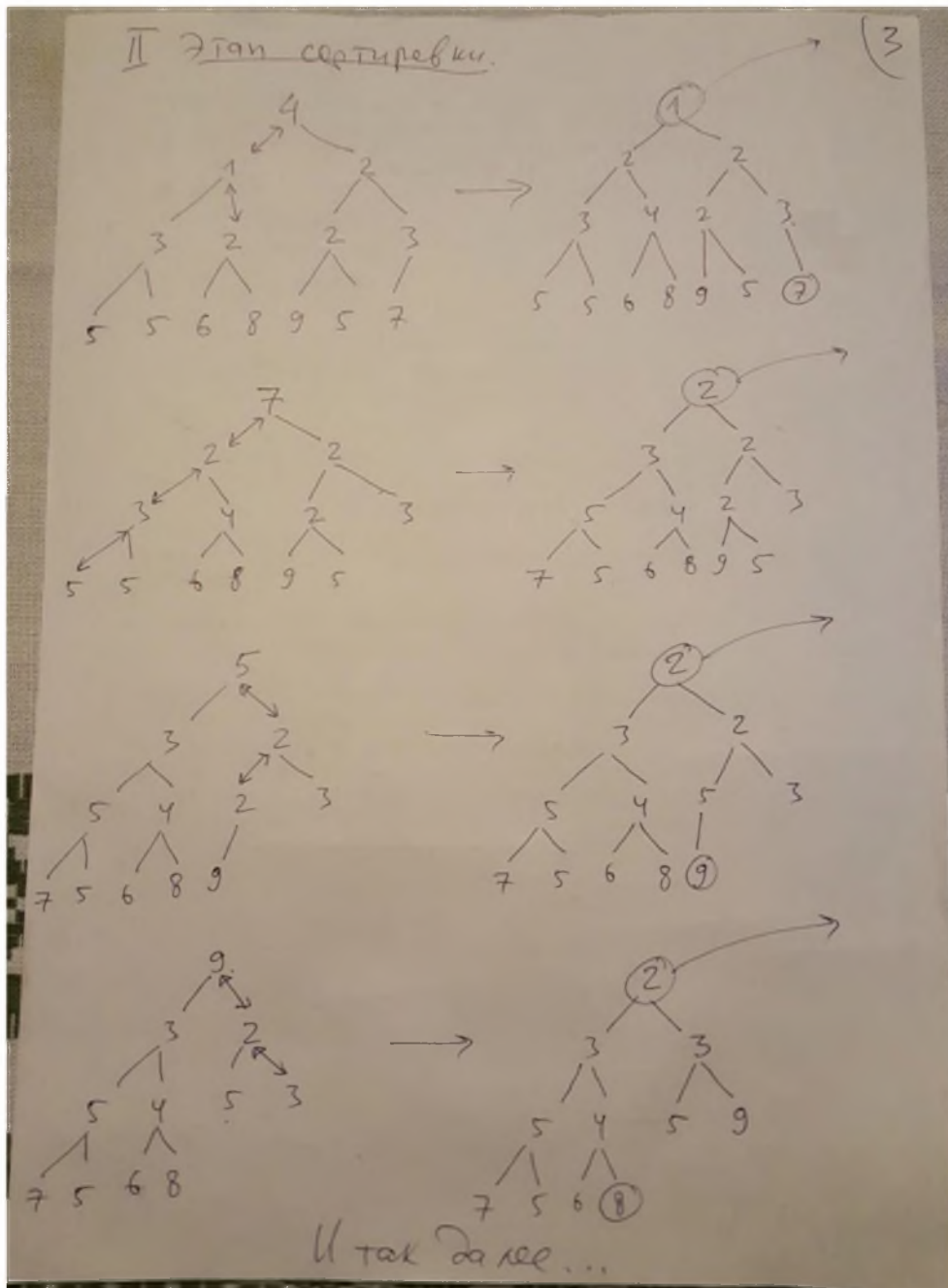
5)  $i = 2$ .  $A_2 \leq \min \{ A_5, A_6 \} ?$  нет.  
 (7) (2) (2)

$A = \{ 3, 5, 2, 0, 2, 2, 2, 1, 5, 6, 8, 9, 5, 3, 4 \}$



Нормы  
 сбалансированные  
 дерево

- $A_6 = \min \{ A_{13}, A_{14} \} ? \text{ yes!}$
- $A = \{ 3 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_1 = \min \{ A_2, A_3, A_4 \} ? \text{ yes.}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_1 = \min \{ A_2, A_3, A_4 \} ? \text{ yes.}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_3 = \min \{ A_7, A_8 \} ? \text{ yes}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_0 = \min \{ A_1, A_2 \} ? \text{ yes}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_1 = \min \{ A_2, A_3, A_4 \} ? \text{ yes.}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_3 = \min \{ A_7, A_8 \} ? \text{ yes}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_1 = \min \{ A_2, A_3, A_4 \} ? \text{ yes.}$
- $A = \{ 3 \rightarrow 0 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 7 \rightarrow 4 \}$   
 $A_3 = \min \{ A_7, A_8 \} ? \text{ yes}$



7. **Поразрядная сортировка**  
 Автор – Герман Холлерит (1860-1929).

Применяется для сортировки целых чисел и строк.  
Элементы массива необходимо разделить на разряды. Например, слово можно разделить по буквам, число - по цифрам...

До сортировки необходимо знать два параметра:  $k$  и  $m$ , где  
 $k$  - количество разрядов в самом длинном элементе,  
 $m$  - разрядность элементов: количество возможных значений разряда элемента.

При сортировке русских слов  $m = 33$ , так как буква может принимать не более 33 значений. Если в самом длинном слове 10 букв,  $m = 10$ .

Аналогично, для шестнадцатиричных чисел  $m=16$ , если в качестве разряда брать цифру, и  $m=256$ , если использовать побайтовое деление.

Эти параметры нельзя изменять в процессе работы алгоритма.  
Предположим [15], что элементы линейного списка  $L$  есть  $k$ -разрядные десятичные числа, разрядность максимального числа известна заранее.  
Обозначим  $d(j,n)$  -  $j$ -ю справа цифра числа  $n$ , которую можно выразить как

$$d(j,n) = [ n / 10^{j-1} ] \% 10$$

Пусть  $L_0, L_1, \dots, L_9$  - вспомогательные списки (карманы), вначале пустые.  
Поразрядная сортировка состоит из двух процессов, называемых распределение и сборка и выполняемых для  $j=1,2,\dots,k$ .

Фаза распределения разносит элементы  $L$  по карманам: элементы  $l_i$  списка  $L$  последовательно добавляются в списки  $L_m$ , где  $m = d(j, l_i)$ . Таким образом получаем десять списков, в каждом из которых  $j$ -тые разряды чисел одинаковы и равны  $m$ .

Фаза сборки состоит в объединении списков  $L_0, L_1, \dots, L_9$  в общий список  
 $L = L_0 \Rightarrow L_1 \Rightarrow L_2 \Rightarrow \dots \Rightarrow L_9$

**Пример.** Рассмотрим работу алгоритма на входном списке  
 $0 \Rightarrow 8 \Rightarrow 12 \Rightarrow 56 \Rightarrow 7 \Rightarrow 26 \Rightarrow 44 \Rightarrow 97 \Rightarrow 2 \Rightarrow 37 \Rightarrow 4 \Rightarrow 3 \Rightarrow 3 \Rightarrow 45 \Rightarrow 10$ .

Максимальное число содержит две цифры, значит, разрядность данных  $k=2$ ,  
 $m=10$ .

Первый проход,  $j=1$ .

Распределение по первой справа цифре:

$L_0: 0 \Rightarrow 10$  // все числа с первой справа цифрой 0

$L_1:$  пусто

$L_2: 12 \Rightarrow 2$

L3: 3 => 3  
L4: 44 => 4  
L5: 45  
L6: 56 => 26  
L7: 7 => 97 => 37  
L8: 8  
L9: пусто // все числа с первой справа цифрой 9

Сборка:  
соединяем списки  $L_i$  один за другим  
L: 0 => 10 => 12 => 2 => 3 => 3 => 44 => 4 => 45 => 56 => 26 => 7 => 97 => 37 => 8

Второй проход,  $j=2$ .  
Распределение по второй справа цифре:  
L0: 0 => 2 => 3 => 3 => 4 => 7 => 8  
L1: 10 => 12  
L2: 26  
L3: 37  
L4: 44 => 45  
L5: 56  
L6: пусто  
L7: пусто  
L8: пусто  
L9: 97

Сборка:  
соединяем списки  $L_i$  один за другим  
L: 0 => 2 => 3 => 3 => 4 => 7 => 8 => 10 => 12 => 26 => 37 => 44 => 45 => 56 => 97

Особенности алгоритма:

1. Имеется два варианта поразрядной сортировки LSD (least significant digit — сначала младший разряд) и MSD (most significant digit — сначала старший разряд)
2. Сортировка является устойчивой.
3. Вычислительная сложность  $O(N)$ .

## 8. \*Сортировка с помощью красно-черного дерева

Красно-чёрное дерево (англ. Red-black tree, RB-Tree) — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла

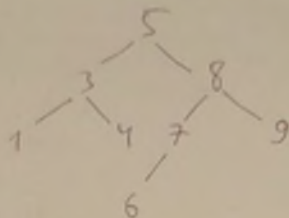
дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».

Изобретателем красно-чёрного дерева считают немца Рудольфа Байера (род. 1939г.).

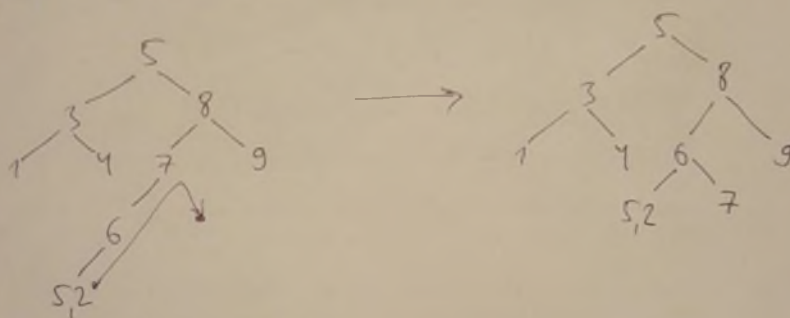
Пример преобразований RBTtree

Пример (предрабочее RB-tree). (1)

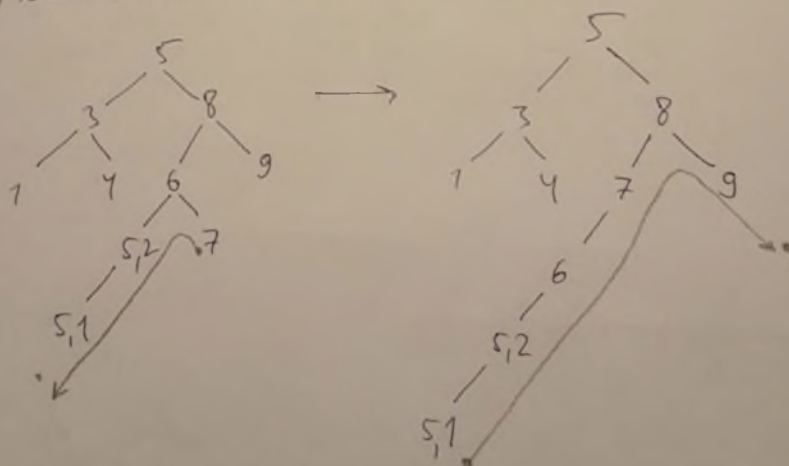
1) Ищем RBT



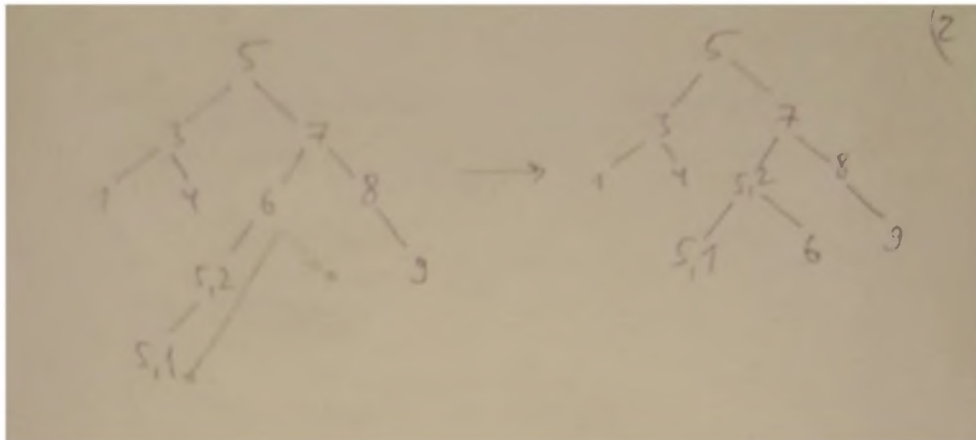
2) Добавим 5,2



3) Добавим 5,1







**Лабораторная работа №3**  
**Арифметика рациональных чисел**

1. Рациональные числа состоят из числителя и знаменателя в виде неограниченных целых чисел.
2. Реализовать арифметические операции.
3. Реализовать алгоритм Евклида нахождения наибольшего общего делителя для приведения дроби к несократимому виду.
4. Реализовать алгоритм преобразования обыкновенной дроби  $m/n$  в периодическую  $a,b(c)$ .
5. Реализовать преобразование периодической дроби в обыкновенную.

**Лабораторная работа №4**  
**Рекурсивная функция вычисления определителя**

1. Написать функцию вычисления определителя квадратной матрицы, раскрывая определитель по строке.
2. Вычислить определитель именной матрицы Редхеффера. Ответ сравнить с теоретическими результатами.
3. Решить квадратную систему линейных уравнений методом Крамера. Функция выдает ответы:
  - решение в случае его единственности;
  - нет решений;
  - решений бесконечно много.
4. Определить сложность алгоритма.

**Лабораторная работа №5**  
**Алгебраический алгоритм вычисления обратной матрицы**

1. Написать функцию вычисления определителя квадратной матрицы, раскрывая определитель по строке.

2. Написать функцию вычисления обратной матрицы, с использованием алгебраических дополнений.
3. Написать функцию умножения матрицы на вектор.
4. Решить квадратную систему линейных уравнений методом обратной матрицы.  
Функция выдает ответы:
  - решение в случае его единственности;
  - нет решений или решений бесконечно много.
5. Определить сложность алгоритма.

#### **Лабораторная работа №6**

##### **Вычислительный алгоритм решения системы линейных уравнений**

1. Написать функцию решения прямоугольной системы линейных уравнений методом Гаусса.  
Функция выдает ответы:
  - решение в случае его единственности;
  - нет решений;
  - если решений бесконечно много, получить общее решение системы, как сумму частного решения и линейной комбинации базисных решений.
2. Определить сложность алгоритма.

#### **Лабораторная работа №7**

##### **Одномерная оптимизация с помощью алгоритма золотого сечения**

1. Создать конфигурационную структуру с полями:
  - Диапазон для выбора начальной точки;
  - Количество выборов начальной точки;
  - Предельное количество итераций при поиске интервала локализации;
  - Начальный шаг при поиске интервала локализации;
  - Задача на минимум или на максимум;
  - Значение золотого числа;
  - Максимальное допустимое нарушение золотого сечения в золотой тройке;
  - Точность экстремальной точки;
  - Расстояние различимости двух близких экстремумов;
  - Условие сходимости/расходимости при поиске интервала локализации;
2. Реализовать алгоритм поиска интервала локализации.
3. Реализовать алгоритм уменьшения интервала локализации.
4. Реализовать тестирование функций.

#### **Лабораторная работа №8**

##### **Дифференциальные конструкции функции нескольких переменных**

## Лабораторная работа №17

### Нечеткие конструкции

#### Задание 1. Операции с нечеткими множествами

$N$  - номер варианта (порядковый номер в списке группы).

Заданы три нечетких множества с кусочно-линейными функциями принадлежности. Вершины этих функций имеют координаты:

A: (2; 0), (3; 1), (N+3;1), (N+5; 0),

B: (1; 0), (2; 1), (N+3;0),

C: (0; 0), (N; 1), (N+4; 0).

Найдите нечеткие множества:

$D = (A \cap B) \cup C$

$E = A \setminus (B \cap C)$ .

#### Задание 2. Вычисление релевантности результатов запроса с помощью нечетких множеств

Для брачного агентства необходимо создать модуль ранжирования результатов запроса Зклиентки. При формировании запроса информационная система предлагает клиентке два критерия для кандидатов: зар.плата и возраст.

Варианты значения (или термы) критериев (или лингвистических переменных):

- низкая зар.плата, хорошая зар.плата, высокая зар.плата;
- молодой, средний возраст, пожилой.

Для обеих лингвистических переменных (зар.плата, возраст )задайте соответствующие термы. Каждый критерий клиентка сама задает в виде нечеткого множества.

База данных агентства содержит три записи:

- Иванов, зар.плата 30-N тыс. руб, 40 – 2\*N лет,
- Петров, зар.плата 15+N тыс. руб, 30 – \*N лет,
- Сидоров, зар.плата 20+5N тыс. руб, 30 + 2\*N лет,

Проведите ранжирование потенциальных кандидатов для следующих запросов:

Z1 = «Женщина ищет мужчину с хорошей зар.платой И среднего возраста»,

Z2 = «Женщина ищет молодого мужчину ИЛИ с высокой зар.платой».

#### Задание 3. Применение нечетких чисел при реализации ИИС

Факторы ИИС представлены в виде нечетких чисел:

- средняя норма ежедневных трудозатрат  $H = [5; 0,2; 0,1]$ (час/день),
- количество рабочих дней в месяце  $D = [20; 1; 2]$ .
- средний размер заработной платы  $Z = [15 + 2*N; 0,1; 0,3]$  (тыс.руб).

Вычислите признак: производительность труда  $P$  в виде нечеткого множества.

#### Задание 4. Получение новой информации с помощью нечетких отношений

Заданы множества:

X – группа людей из четырех человек,

Y–автомобили N+2 марок,  
Z–стоимость ежегодного обслуживания (дешевое, среднее, дорогое, очень дорогое).

Постройте нечеткие отношения:

U: X ->Y – в какой степени марка автомобиля нравится человеку,

V: Y ->Z – какова стоимость обслуживания данной марки.

Постройте новое отношение:

W: X ->Z – какая стоимость обслуживания ожидает автовладельца.

### **Задание 5. Реализация нечеткого вывода**

Предлагается создать систему нечеткого вывода для управления автомобилем роботом.

Составьте лингвистические переменные с базовыми термами:

1. Скорость автомобиля. Термы: «Стоит», «Малая», «Средняя», «Высокая», «Очень высокая»

2. Расстояние до впереди идущего автомобиля. Термы: «Малое», «Среднее», «Большое», «Очень большое».

3. Управление одной педалью «Тормоз/Газ», от -100% до 100%. Термы: «Полный тормоз» (-100%), «Полный газ» (100%), другие термы составьте по своему усмотрению.

Постройте 10 правил нечеткого вывода. Например,

«Если скорость робота мала, расстояние до впереди идущего автомобиля среднее и скорость этого автомобиля высокая, то сильно газовать».

Используя систему нечеткого вывода, определите величину управления педалью Тормоз/Газ для ситуации:

- скорость робота равна 50км/ч,
- расстояние до следующей машины 100 м.,
- скорость следующей машины 20 км/ч

### **Лабораторная работа №18**

#### **Методы численного интегрирования**

1. Методы правых прямоугольников.
2. Метод центральных прямоугольников.
3. Метод трапеций.
4. Метод парабол (метод Симпсона).
5. Увеличение точности.
6. Метод Гаусса.
7. Метод Гаусса — Кронрода.
8. Метод Чебышёва.
9. Интегрирование при бесконечных пределах.

### Лабораторная работа №19

#### Численные методы решения дифференциальных уравнений и систем

1. Метод Эйлера.
2. Методы Рунге-Кутты 2, 3, 4 порядков.

### Лабораторная работа №20

#### Геометрические алгоритмы

1. Найти выпуклую оболочку множества точек на плоскости.
2. Алгоритм вычисления площади многоугольника.
3. Принадлежность точки многоугольнику.
4. Реализовать иерархию геометрических примитивов для визуализации движения составных фигур.
5. Реализовать алгоритм освещения трехмерной фигуры.

### Лабораторная работа №21

#### Эволюционные алгоритмы

Создание генетического алгоритма, выбор кодирования, кроссовера, схемы мутации и т.п.

**Задание.** Задана функция положительная  $f(x)$  одной переменной на отрезке  $[a, b]$ . Найти минимум функции. Отрезок  $[a, b]$  разбить сеткой из  $2^5 = 32$  точек. Особь кодирует узел сетки 5-ю битами.

#### Тестовые вопросы по генетическому алгоритму:

1. Дайте определение генетическому алгоритму. 2. Что такое эвристические алгоритмы? 3. Дайте определение понятиям: «ген», «хромосома», «генотип», «особь» и «фенотип». 4. Дайте понятие функции приспособленности 5. Перечислите основные шаги алгоритма процесса формирования нового поколения. 6. Перечислите операторы генетического алгоритма и их назначение. 7. Опишите операторы селекции. 8. Как вычислить вероятность каждой особи в методе селекции с помощью рулетки? 9. Какие операторы скрещивания вам известны? 10. Перечислите известные вам операторы мутации. 11. Какие виды кодирования вам известны? 12. Как декодировать строку, записанную в двоичном коде? 13. Как декодировать строку, записанную в коде Грэя? 14. Как связаны вещественные числа с числами в бинарном коде? 15. Какие из операторов генетического алгоритма выполняются с использованием элементов случайности, а какие по строго детерминированным правилами? 16. Перечислите основные отличия генетических алгоритмов от традиционных методов поиска решений. 17. Опишите схему работы генетического алгоритма. 18. Что может являться критерием остановки работы генетического алгоритма?

#### Литература

1. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии
2. Жамбю М. Иерархический кластер-анализ и соответствия
3. Скворцов Л.М. Численное решение обыкновенных дифференциальных и дифференциально-алгебраических уравнений
4. Седжвик Р. Фундаментальные алгоритмы на C++
5. Эвристические методы оптимизации. Учебное пособие. Составитель А.А. Мицель. Томск: Томский государственный университет систем управления и радиоэлектроники. – 2019. – 73 с.
6. Лавров Н.Г. Конспект лекций по нечетким множествам.
7. Леоненков А. В. Нечеткое моделирование в среде MATLAB и fuzzyTECH. СПб.: БХВ, 2005. 736 с.
8. Моделирование сложных вероятностных систем : учеб. пособие / В. Г. Лисиенко, О. Г. Трофимова, С. П. Трофимов, Н. Г. Дружинина, П.А. Дюгай. Екатеринбург: УРФУ, 2011. 200 с.
9. Сборник олимпиадных задач для специальности «Вычислительные машины, комплексы, системы и сети» / В.П.Битюцкий, С.Л.Гольштейн, С.В.Поршнев, С.И.Рогович, С.С.Соколов, С.П.Трофимов, А.В.Цветков. Учебное пособие. Допущено учебно-методическим объединением вузов по университетскому образованию в качестве учебного пособия для студентов вузов, обучающихся по специальности 230101 – ВМКСС. М.: КНОРУС, 2010. 280 с.
10. Калугина Д. Проверка статистических гипотез. Отчет по лабораторной работе
11. Knuth, D. E. The Art of Computer Programming. — Addison-Wesley, 2005.
12. Федоряева Т.И. Комбинаторные алгоритмы. Учебное пособие. Новосибирск. 2011. 118с.
13. Мичурин А. <http://www.michurin.net/>
14. Dresegor. RSA: от простых чисел до электронной подписи <https://habr.com/ru/post/534014/>
15. Поразрядная сортировка. // Analogist.manual.ru, [http://algotlist.ru/sort/radix\\_sort.php](http://algotlist.ru/sort/radix_sort.php)

### Тематика лаб.работ

№	Тема лаб.работы	Вариант	ЛР №1	ЛР№2	№ студента, лаб. работа №3	№ студента, лаб. работа №4
1	Целочисленная арифметика произвольной точности и RSA-шифрование		все			
2	Алгоритмы сортировки строк			все		
3	Арифметика рациональных чисел					
4	Рекурсивная функция вычисления определителя					
5	Алгебраический алгоритм вычисления обратной матрицы					
6	Вычислительный алгоритм решения системы линейных уравнений					
7	Одномерная оптимизация с помощью алгоритма золотого сечения					
8	Дифференциальные конструкции функции нескольких переменных					
9	Алгоритмы нахождения комплексных корней многочленов					182
10	Решение задачи о рюкзаке с помощью переборного комбинаторного алгоритма					
11	Решение задачи о назначениях с помощью переборного комбинаторного алгоритма					
12	Решение задачи коммивояжера с помощью переборного комбинаторного алгоритма					
13	Символьная арифметика					11, 48, 94,99,108
14	Вероятностные алгоритмы					
15	Проверка статистических гипотез					
16	Кластерный анализ					
17	Нечеткие конструкции	Операции с нечеткими множествами				
		Вычисление релевантности результатов запроса с				

		помощью нечетких множеств				
		Применение нечетких чисел при реализации ИИС				
		Получение новой информации с помощью нечетких отношений				
		Реализация нечеткого вывода				
18	<b>Методы численного интегрирования</b>	Методы правых прямоугольников.				
		Метод центральных прямоугольников.				
		Метод трапеций.				
		Метод парабол (метод Симпсона).				
		Увеличение точности.				
		Метод Гаусса.				
		Метод Гаусса — Кронрода.				
		Метод Чебышёва.				
		Интегрирование при бесконечных пределах.				
19	<b>Численные методы решения дифференциальных уравнений и систем</b>	Метод Эйлера.				
		Методы Рунге-Кутты 2 порядка				
		Методы Рунге-Кутты 3 порядка				
		Методы Рунге-Кутты 4 порядка				
20	<b>Геометрические алгоритмы</b>	Найти выпуклую оболочку множества точек на плоскости.				
		Алгоритм вычисления площади многоугольника.				
		Принадлежность точки многоугольнику.				
		Реализовать иерархию геометрических примитивов для визуализации движения составных фигур.				



		Реализовать алгоритм освещения трехмерной фигуры.				
21	Эволюционные алгоритмы	Генетический алгоритм одномерной минимизации				

## Контрольная работа по дисциплине «Алгоритмы и анализ сложности»

$V$  – номер варианта, равен порядковому номеру студента в списке команды

1. Массовая задача состоит из  $V$  миллионов серий, в каждой из которых по  $V$  миллиардов индивидуальных задач. Назовите словами количество индивидуальных задач в массовой задаче.
2. Определите наиболее информативную асимптотику функции сложности алгоритма  $P_{if}(n) = (V*n - \log n)^2$
3. Выполните сортировку массива  $X = \{3, 4, V, 6, 5, 2, 1, 2*V, 10, 3\}$  по возрастанию тремя методами  $V\%8 + 1$ ,  $(V+3)\%8 + 1$ ,  $(V+6)\%8 + 1$  из перечисленных ниже:

- 3.1. Сортировка пузырьком BubbleSort
- 3.2. Сортировка вставками InsertionSort
- 3.3. Сортировка слиянием MergeSort
- 3.4. Поразрядная сортировка RadixSort
- 3.5. Быстрая сортировка QSort
- 3.6. Сортировка деревом SortTree
- 3.7. Пирамидальная сортировка HeapSort
- 3.8. Красно-черное дерево RBTree

4. Найдите минимальный путь из вершины  $A$  в вершину  $B$  для заданной матрицы расстояний. Примените алгоритм Дейкстры.

	A	B	C	D	E	F
A			V		6	3
B	1		1	5		
C	2				3	
D	4	1	5			
E	3	V		1		2
F	V	7		3	V	

5. С помощью алгоритма динамического программирования найдите размен суммы  $S = 25$  руб. монетами по 2, 5, 7 руб. с минимальным количеством монет.
6. Найдите первую производную функции

$$f(x) = \sqrt{\sin(Vx)}$$

в точке  $x=1$ . Используйте алгоритм численного дифференцирования с оптимальным шагом. Определите абсолютную погрешность.

7. Случайная величина задана плотностью распределения

$$f(x) = \begin{cases} 0, & \text{если } x < 0 \\ C * (1 - \frac{x}{V}), & \text{если } 0 \leq x \leq V \\ 0, & \text{если } x > V \end{cases}$$

Определите константу  $C$  и постройте генератор этой величины с помощью метода «обратной» функции.

**Перечень теоретических вопросов и заданий для итогового экзамена по дисциплине «Алгоритмы и анализ сложности»**

1. Понятие функции сложности алгоритма. Асимптотическая оценка сложности. Три вида асимптотических оценок. Что означают записи  $P(n)=O(n)$ ,  $P(n)=\Omega(n)$ ,  $P(n)=\Theta(n)$ ? Что означает фраза « $f(n)$  асимптотически меньше  $g(n)$ »? Виды семейств функций сложности. Полиномиальные и экспоненциальные алгоритмы.
2. Алгоритм сортировки пузырьком BubbleSort. Порядок сложности алгоритма. Особенности метода.
3. Алгоритм быстрой сортировки QSort. Порядок сложности алгоритма. Особенности метода.
4. Оптимальные алгоритмы сортировки. Теорема (с доказательством) об оптимальной сложности алгоритма сортировки с использованием операции if.
5. Рекурсивные функции. Кратность рекурсии. Графическое представление схемы стека вызовов функций. Рекурсивные функции и ресурсы компьютера: время и память. Что помещается в стек при вызове функции?
6. Алгоритм сортировки СортДерево. Порядок сложности алгоритма. Особенности метода.
7. Алгоритм сортировки слиянием MergeSort. Порядок сложности алгоритма. Особенности метода.
8. Алгоритм сортировки вставками InsertionSort. Порядок сложности алгоритма. Особенности метода.
9. Алгоритм поразрядной сортировки вставками RadixSort. Порядок сложности алгоритма. Особенности метода.
10. Алгоритм пирамидальной сортировки HeapSort. Порядок сложности алгоритма. Особенности метода
11. Алгоритм сортировки красно-черным деревом RBTree. Порядок сложности алгоритма. Особенности метода
12. Греческий алфавит. Названия больших чисел с двоичным и десятичным основаниями. Основная эквиваленция для эквивалентной переформулирования теорем.
13. Алгоритм генерации перестановок в лексикографическом порядке. Сложность алгоритма.
14. Умножение многочленов. Быстрый алгоритм умножения многочленов.
15. Быстрый алгоритм Карацубы умножения длинных чисел.
16. Контейнеры. Виды контейнеров. Структура контейнеров. Использование итераторов для перебора элементов контейнера.
17. Виды оптимальных задач на графах. Метод динамического программирования для решения дискретных оптимальных задач на графах. Основной принцип динамического программирования.

18. Нахождение кратчайшего пути в графе методом динамического программирования. Прямой и обратный ход алгоритма. Уравнение Беллмана для данной задачи.
19. Задача о минимальном размене. Решение методом динамического программирования. Прямой и обратный ход алгоритма. Уравнение Беллмана для данной задачи.
20. Структура xml-формата данных. Виды узлов. Структура узлового элемента.
21. Библиотека классов, например на C#, для открытия xml-файла, просмотра и изменения.
22. Методы численного дифференцирования. Вычисление первой и второй производных для функции одной переменной. Вычисление частных и смешанных производных для функции нескольких переменных. Точность методов дифференцирования.
23. Методы численного интегрирования. Точность методов.
24. Представление вещественных чисел в компьютере. Нормализованная форма. Мантисса и порядок вещественного числа. Машинный ноль, машинный эpsilon, машинная бесконечность, «правый сосед» числа. Машинная вещественная ось.
25. Одномерная случайная величина. Функциональные характеристики СВ: функция плотности и функция распределения. Интегрально-дифференциальная связь. Алгоритм генерации СВ методом «обратной функции»
26. Алгоритмы анализа одномерных данных. Представительная выборка. Построение и проверка статистических гипотез при анализе данных. Понятия статистики, статистической гипотезы. Критерий проверки статистической гипотезы. Описать критерий Пирсона для проверки гипотезы о совпадении выборочного и теоретического распределения.
27. Основные понятия теории графов. Вершины. Ребра. Компоненты связности. Цикл. Цепь. Способы задания графов. Матрица смежностей. Списки смежностей.
28. Алгоритм RSA шифрования с открыты ключом. Постановка проблемы. Роли Алисы, Боба и хакера. Понятие односторонней функции. Обоснование криптостойкости алгоритма.
29. Теоретико-числовые алгоритмы. Алгоритм Эвклида нахождения НОД двух целых чисел. Обобщенный алгоритм Эвклида решения диофантового уравнения  $a*x+b*y=d$ .
30. Модулярная арифметика. Сложение, умножение, обратный элемент по модулю. Алгоритм вычисления обратного элемента по модулю обобщенным алгоритмом Эвклида.
31. Китайская теорема об остатках. Алгоритм Гарнера решения системы уравнений с остатками.
32. Алгоритм перемешивания массива, улучшающий асимптотические свойства алгоритмов сортировки.

33. Двоичное кодирование целых чисел кодами Грея. Достоинства кодов Грея перед кодированием двоичной системой счисления.
34. Эволюционные алгоритмы. Генетический алгоритм. Понятия: ген, аллель, хромосома, особь, популяция. Этапы генетического алгоритма: Селекция, Скрещивание, Мутация, Отбор. Функция приспособленности. Решение задачи одномерной минимизации генетическим алгоритмом.
35. Быстрые алгоритмы. Быстрое умножение матриц.
36. Тестирование и отладка алгоритма. Тестирование алгоритма и демонстрация алгоритма – разный исходный код! Требования к тестированию. Процентные правила хорошего стиля программирования. Способы записи алгоритма. Способы обработки ошибок: вывод сообщений на консоль, запись в log-файл, иерархия исключений.
37. Динамические структуры данных. Структура динамического узла: информационные и служебные поля. Интерфейс динамической структуры данных. Шаблонные классы. Библиотека шаблонных классов STL. Итераторы.
38. Метода наименьших квадратов аппроксимации множества двумерных точек не плоскости параметрическим семейством трендов. Виды семейств трендов: линейные, полиномиальные, обратные, гармонические. Особенности алгоритма. Функция квадратичных невязок. Решение задачи в Excel с помощью сервиса «Поиск решения»
39. Дерево, как способ структурирования данных. Сортировочное и пирамидальное дерево. Сбалансированное и несбалансированное дерево. Негативные особенности несбалансированного дерева.
40. Численные методы решения дифференциального уравнения. Метод Эйлера. Методы Рунге-Кутты 2, 3, 4 порядков.

### Примеры задач

1. Имеется одномерная выборка из 100 элементов, полученная из нормального распределения с параметрами  $m=2$ ,  $\sigma=3$ . Методом Пирсона проверить гипотезу о совпадении выборочного распределения и теоретического нормального распределения с параметрами  $m=2$ ,  $\sigma=4$ . Уровень значимости  $\alpha = 1\%$ . Количество конечных и полубесконечных карманов 12.
2. Массовая задача состоит из  $V$  миллионов серий, в каждой из которых по  $V$  миллиардов индивидуальных задач. Назовите словами количество индивидуальных задач в массовой задаче.
3. Определите наиболее информативную асимптотику функции сложности алгоритма  $P_{if}(n) = (5*n - \log n)^2$
4. Выполните сортировку массива  $X = \{3, 4, V, 6, 5, 2, 1, 2*V, 10, 3\}$  по возрастанию одним из перечисленных ниже методов:

- Сортировка пузырьком BubbleSort
  - Сортировка вставками InsertionSort
  - Сортировка слиянием MergeSort
  - Поразрядная сортировка RadixSort
  - Быстрая сортировка QSort
  - Сортировка деревом SortTree
  - Пирамидальная сортировка HeapSort
  - Красно-черное дерево RBTree
5. Найдите минимальный путь из вершины А в вершину В для заданной матрицы расстояний. Примените алгоритм Дейкстры.

	A	B	C	D	E	F
A			V		6	3
B	1		1	5		
C	2				3	
D	4	1	5			
E	3	V		1		2
F	V	7		3	V	

6. С помощью алгоритма динамического программирования найдите размен суммы  $S = 25$  руб. монетами по 2, 5, 7 руб. с минимальным количеством монет.
7. Найдите численным методом первую и вторую производные функции  $f(x) = \sqrt{\sin(Vx)}$  в точке  $x=1$ . Определите абсолютную погрешность.
8. Найдите численным методом частные производные  $f'_x$  и  $f'_y$  функции  $f(x, y) = x^2 - 2x * y + 3 * y^2$  в точке  $(x_0, y_0)=(1, 2)$ . Определите абсолютную погрешность.
9. Случайная величина задана плотностью распределения
10. 
$$f(x) = \begin{cases} 0, & \text{если } x < 0 \\ C * (1 - \frac{x}{V}), & \text{если } 0 \leq x \leq V \\ 0, & \text{если } x > V \end{cases}$$
11. Определите константу C и постройте генератор этой величины с помощью метода «обратной» функции.

## 12. Рекурсия

Имеется большой одномерный массив A целых чисел, упорядоченных по возрастанию. Напишите рекурсивную функцию *has*, которая определяет, содержится ли целое число  $x$  в этом массиве.

Желательно учесть наличие упорядоченности и использовать деление массива пополам. Также напишите *main* и *test*.

Прототип функции *has*:

*int has( int \*A, int N, int x);*

13. Используя встроенный генератор случайных чисел решить приближенно следующую вероятностную задачу. В урне 4 белых, 6 черных и 10 красных шаров. Вынимаем 2 шара. Найти вероятность, что оба шара разного цвета. Найти теоретическое решение задачи и сравнить. Выполнить 10 миллионов опытов
14. Решить диофантово уравнение в целых числах
$$17*x+83*y=1$$
15. Вычислить обратный по модулю элемент  $73^{-1} \pmod{101}$
16. Выполнить передачу секретного числа  $P=17$  по открытому каналу алгоритмом RSA-шифрования

**Приложение 1.**  
**Пример отчета по лабораторной работе**

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Уральский федеральный университет имени первого Президента России Б.Н.Ельцина»  
Институт радиоэлектроники и информационных технологий – РТФ

**Анализ сложности алгоритмов сортировки строк**

Отчёт по лабораторной работе  
по дисциплине «Алгоритмы и анализ сложности»  
Вариант 6

Выполнил:  
студент группы

Преподаватель:  
доцент, к.ф.-м.н.  
Трофимов С.П.

2023



## СОДЕРЖАНИЕ

<u>Задание</u> .....	50
<u>Теоретическая часть</u> .....	51
<u>Инструкция пользователя</u> .....	53
<u>Инструкция программиста</u> .....	55
<u>Тестирование</u> .....	56
<u>Выводы</u> .....	57
<u>Литература</u> .....	59
<u>Приложение</u> .....	60

## Задание

Реализовать один из алгоритмов сортировки строк:

### 6. Пирамидальная сортировка HeapTree

Выбор алгоритма выбирается по согласованию с преподавателем.

Для алгоритма определить сложность относительно наиболее характерной операции (сравнение, перестановка и др.). Вид функции сложности  $F(n)$  подобрать в соответствии с теорией. Например, для оптимальных алгоритмов  $F(n) = C \cdot n \cdot \log_2(n)$ . Найти также коэффициент пропорциональности  $C$ . Для аппроксимации можно использовать метод наименьших квадратов и сервис «Поиск решения».

План проведения эксперимента с алгоритмом называется массовой задачей. Представьте план в виде xml-файла.

Результаты решения массовой задачи записать в текстовый файл в 2 столбика: длина массива, количество операций. Файл импортировать в Excel. В «шапке» листа указать параметры тренда, вычислить квадратичные невязки и минимизировать их сумму с помощью «Данные-Поиск решения»

## Теоретическая часть

Пирамидальная сортировка (англ. Heapsort, «Сортировка кучей») была предложена Дж. Уильямсом в 1964 году.

Сортировка пирамидой использует бинарное сортирующее дерево. Сортирующее дерево — это такое дерево, у которого выполнены условия:

- 1) Каждый лист имеет глубину либо  $d-1$ , либо  $d$ , где  $d$  — максимальная глубина дерева.
- 2) Значение в любой вершине не больше значения её двух потомков.

Удобная структура данных для сортирующего дерева — такой массив `Array`, что `Array[0]` — элемент в корне, а потомки элемента `Array[i]` являются `Array[2i+1]` и `Array[2i+2]`, где  $i = 0, \dots, n/2-1$ .

Алгоритм сортировки будет состоять из двух основных шагов:

- 1) Выстраиваем элементы массива в виде сортирующего дерева:

$Array[i] \leq Array[2i+1]$ ,

$Array[i] \leq Array[2i+2]$

при  $i = n/2-1, \dots, 0$ .

Для этого сравниваем `Array[i]` с  $\min \{Array[2i+1], Array[2i+2]\}$  и при необходимости переставляем.

Этот шаг требует  $O(n)$  операций.

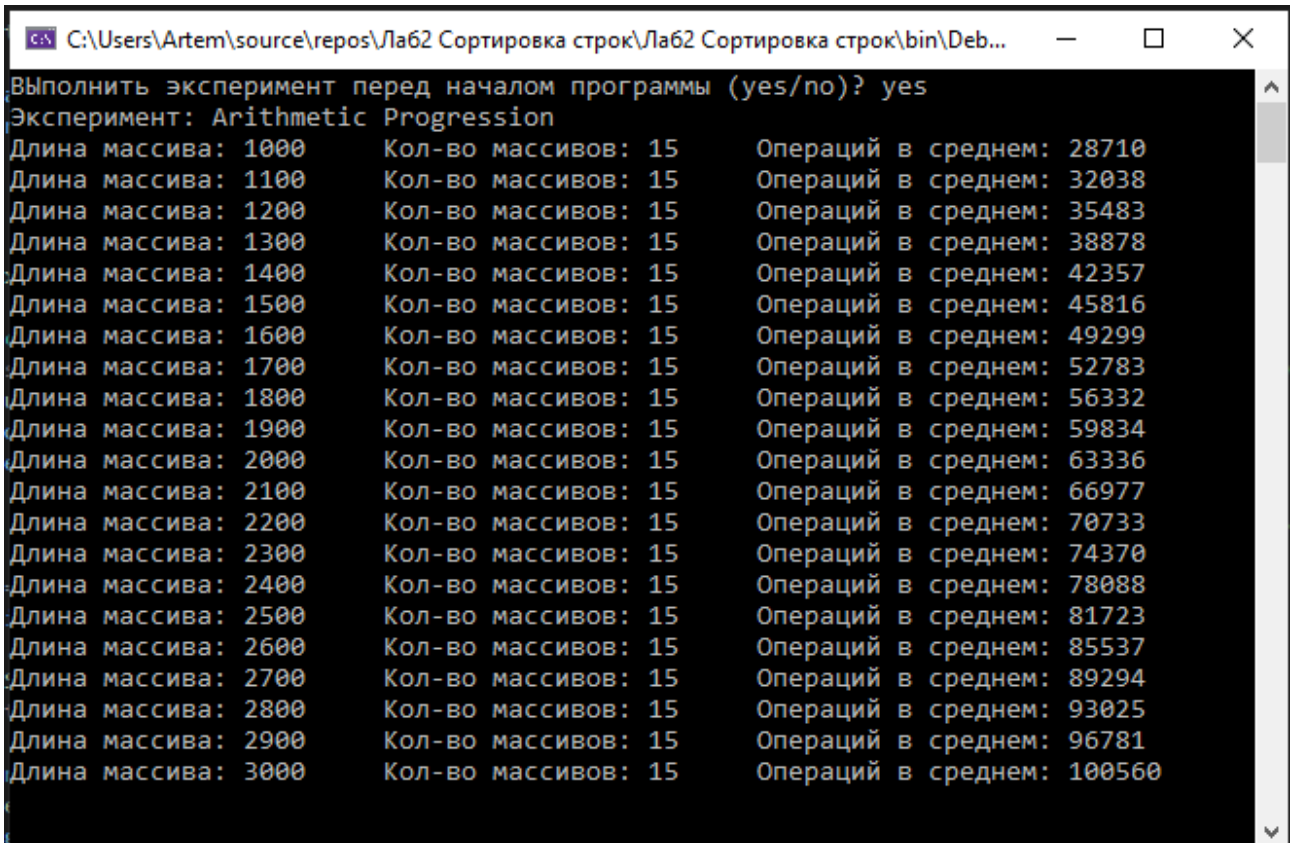
- 2) Будем удалять элементы из корня по одному за раз и перестраивать дерево. То есть на первом шаге обмениваем `Array[0]` и `Array[n-1]`, преобразовываем `Array[0], Array[1], \dots, Array[n-2]` в сортирующее дерево. Затем переставляем `Array[0]` и `Array[n-2]`, преобразовываем `Array[0], Array[1], \dots, Array[n-3]` в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда `Array[0], Array[1], \dots, Array[n-1]` — упорядоченная последовательность. Этот шаг требует  $O(n \cdot \log n)$  операций.

Особенности алгоритма:

- 1) Работает в худшем, в среднем и в лучшем случае (то есть гарантированно) за  $\Theta(n \log n)$  операций при сортировке  $n$  элементов.
- 2) Количество применяемой служебной памяти не зависит от размера массива (то есть,  $O(1)$ ).
- 3) Из-за сложности алгоритма выигрыш получается только на больших  $n$
- 4) Неустойчив

## Инструкция пользователя

При запуске программы открывается консоль, где пользователю предлагается перед сортировкой собственного массива провести эксперимент и оценить производительность алгоритма. Если он согласен, то на консоль будут выведены результаты эксперимента.



```
C:\Users\Artem\source\repos\Лаб2 Сортировка строк\Лаб2 Сортировка строк\bin\Deb...
Выполнить эксперимент перед началом программы (yes/no)? yes
Эксперимент: Arithmetic Progression
Длина массива: 1000      Кол-во массивов: 15      Операций в среднем: 28710
Длина массива: 1100      Кол-во массивов: 15      Операций в среднем: 32038
Длина массива: 1200      Кол-во массивов: 15      Операций в среднем: 35483
Длина массива: 1300      Кол-во массивов: 15      Операций в среднем: 38878
Длина массива: 1400      Кол-во массивов: 15      Операций в среднем: 42357
Длина массива: 1500      Кол-во массивов: 15      Операций в среднем: 45816
Длина массива: 1600      Кол-во массивов: 15      Операций в среднем: 49299
Длина массива: 1700      Кол-во массивов: 15      Операций в среднем: 52783
Длина массива: 1800      Кол-во массивов: 15      Операций в среднем: 56332
Длина массива: 1900      Кол-во массивов: 15      Операций в среднем: 59834
Длина массива: 2000      Кол-во массивов: 15      Операций в среднем: 63336
Длина массива: 2100      Кол-во массивов: 15      Операций в среднем: 66977
Длина массива: 2200      Кол-во массивов: 15      Операций в среднем: 70733
Длина массива: 2300      Кол-во массивов: 15      Операций в среднем: 74370
Длина массива: 2400      Кол-во массивов: 15      Операций в среднем: 78088
Длина массива: 2500      Кол-во массивов: 15      Операций в среднем: 81723
Длина массива: 2600      Кол-во массивов: 15      Операций в среднем: 85537
Длина массива: 2700      Кол-во массивов: 15      Операций в среднем: 89294
Длина массива: 2800      Кол-во массивов: 15      Операций в среднем: 93025
Длина массива: 2900      Кол-во массивов: 15      Операций в среднем: 96781
Длина массива: 3000      Кол-во массивов: 15      Операций в среднем: 100560
```

Рисунок 1 – Проведение эксперимента

Далее, независимо от того, производился опыт или нет, пользователю предлагается ввести собственный массив строк, разделяя его элементы пробелом. После ввода на консоль будет выведен уже отсортированный массив.

```
C:\Users\Artem\source\repos\Лаб2 Сортировка строк\Лаб2 Сортировка строк\bin\Deb...
Эксперимент: Geometric Progression
Длина массива: 10      Кол-во массивов: 15      Операций в среднем: 95
Длина массива: 20      Кол-во массивов: 15      Операций в среднем: 247
Длина массива: 40      Кол-во массивов: 15      Операций в среднем: 588
Длина массива: 80      Кол-во массивов: 15      Операций в среднем: 1396
Длина массива: 160     Кол-во массивов: 15      Операций в среднем: 3255
Длина массива: 320     Кол-во массивов: 15      Операций в среднем: 7408
Длина массива: 640     Кол-во массивов: 15      Операций в среднем: 16608
Длина массива: 1280    Кол-во массивов: 15      Операций в среднем: 36816
Длина массива: 2560    Кол-во массивов: 15      Операций в среднем: 80779
Длина массива: 5120    Кол-во массивов: 15      Операций в среднем: 176150

Эксперимент: Geometric Progression
Длина массива: 10      Кол-во массивов: 15      Операций в среднем: 100
Длина массива: 50      Кол-во массивов: 15      Операций в среднем: 807
Длина массива: 250     Кол-во массивов: 15      Операций в среднем: 5684
Длина массива: 1250    Кол-во массивов: 15      Операций в среднем: 37208
Длина массива: 6250    Кол-во массивов: 15      Операций в среднем: 229602

Введите строки для сортировки, разделяя их пробелом: fde abc aan liv kjh evc
Отсортированный массив: aan abc evc fde kjh liv _
```

Рисунок 2 – Сортировка пользовательского массива строк

## Инструкция программиста

В программе, написанной на языке C#, алгоритм пирамидальной сортировки представлен в классе Program. Для его реализации была написана функция HeapSort:

```
public static int HeapSort(string[] array)
```

принимая массив строк и сортирующая его алгоритмом HeapSort. Для возможности оценивать производительность сортировки функция возвращает значение счётчика той операций, которая вызывалась чаще всего.

Для удобства самая частая операция в HeapSort вынесена в отдельную функцию Heapify:

```
private static void Heapify(string[] array, int heapSize, int i, ref int swapsCount, ref int branchesCount)
```

принимая массив строк, размер формируемой двоичной кучи, индекс корневого узла и ссылки на счетчики операций. Данная функция используется для формирования из массива строк бинарного дерева и вызывается как в HeapSort, так и рекурсивно.

Помимо этих основных функций, в программе реализованы вспомогательные. Это функция PrintArray:

```
private static void PrintArray(string[] array)
```

принимая массив строк и выводящая его на консоль, а также функции GetExperimentResults:

```
private static void GetExperimentResults()
```

выполняющая экспериментальные вычисления и выводящая их в консоль, и GetOperationsCount:

```
private static int GetOperationsCount(int minElement, int maxElement, int repeatsCount, int arrayLength)
```

принимая параметры для создания массивов строк (их длину, диапазон значений элементов, количество массивов); она формирует массивы, запускает функцию сортировки для них и возвращает счётчик самой часто вызываемой операции.

## Тестирование

Кроме основных и вспомогательных, программа также содержит функцию Test. Она содержит тесты, проверяющий корректность работы алгоритма сортировки и выполняется в момент запуска программы. Если один или несколько тестов завершились неудачно, на консоль будут выведены соответствующие сообщения, и программа завершит работу. Часть кода тестов представлена ниже, с полным кодом можно ознакомиться в Приложении.

```
private static bool Test() // Тестовая функция
{
    var isAllTestsCompleted = true;
    {
        var startArray = new[] { "bdg", "avc", "hdf", "aa", "dg", "chd" };
        var expectedArray = new[] { "aa", "avc", "bdg", "chd", "dg", "hdf" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 1 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "enfgh", "sgasf", "asdg", "ksd", "asa", "bhj", "ksgd", "assa", "bhdfj" };
        var expectedArray = new[] { "asa", "asdg", "assa", "bhdfj", "bhj", "enfgh", "ksd", "ksgd", "sgasf" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 2 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "mbdsdfg", "avghssdc", "hjsdf", "abfga", "dhjsg", "lnvchd" };
        var expectedArray = new[] { "abfga", "avghssdc", "dhjsg", "hjsdf", "lnvchd", "mbdsdfg" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 3 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
}
```

Рисунок 3 – Часть кода тестирующей функции Test

Помимо тестирования, программа предоставляет возможность провести эксперимент с пирамидальной сортировкой на больших массивах данных. Для этого в xml-файле Experiment был создан план эксперимента. Он включает в себя несколько элементов nodes, каждый из которых описывает свою часть эксперимента: какое количество массивов будет сгенерировано, какой длины, с какими значениями и по какому принципу эти массивы будут изменяться



(здесь реализовано изменения длины массива в арифметической и геометрической прогрессиях).

Данный документ обрабатывается в функции `GetExperimentResults`: значения атрибутов каждого элемента эксперимента будут получены и использованы для генерации массивов (все значения их элементов выбираются случайным образом из заданного диапазона) и последующей сортировки.

Полученные результаты (длины массивов и кол-во операций для их сортировки) заносятся в таблицу Excel. В дополнение к ним рассчитываем также значения тренда для каждого случая и строим графики функции сложности. Скорее всего, они будут отстоять далеко друг от друга, поэтому для их сближения применим метод наименьших квадратов: рассчитываем для каждого случая квадратичные невязки, а затем получим коэффициент для функции сложности с помощью сервиса Excel «Поиск решения».

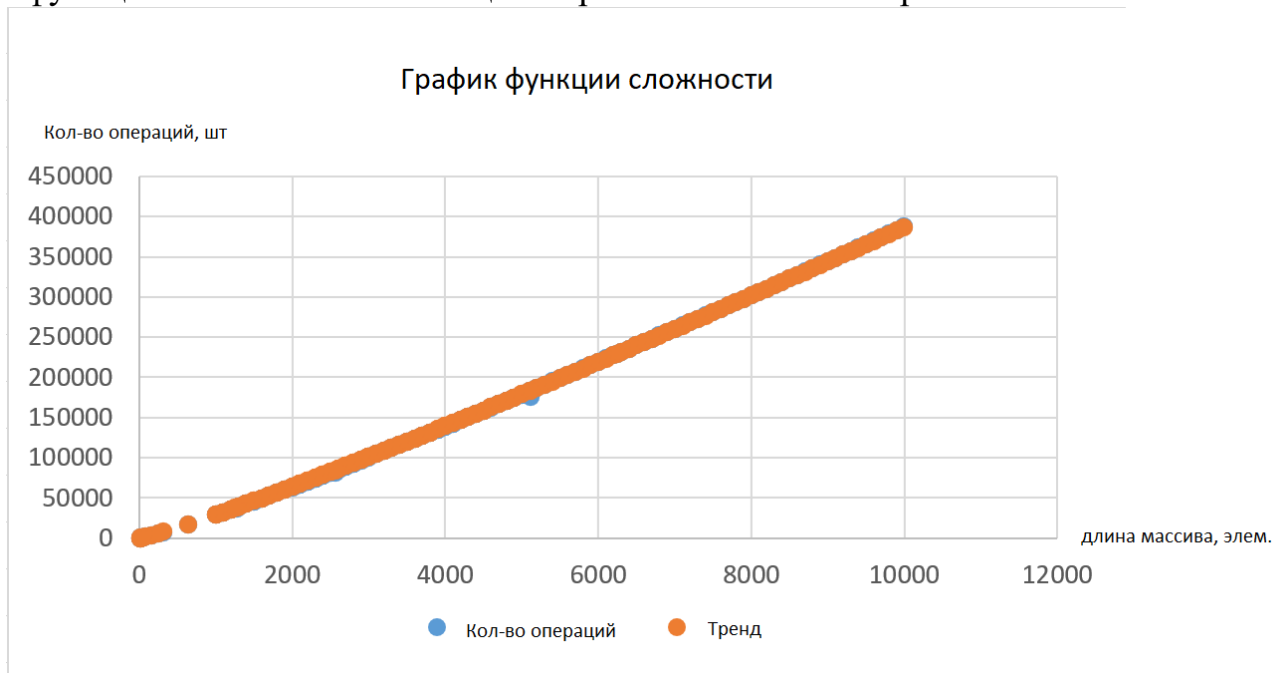


Рисунок 4 – Графики функции сложности после применения сервиса «Поиск решения»

В нашем случае для пирамидальной сортировки строк мы получаем коэффициент пропорциональности  $C = 2,91049832015149 \approx 2,91$ , а функция сложности для данного алгоритма принимает вид  $F(n) = 2,91 * n * \log(n)$ .

## Выводы

В данной работе мы познакомились с одним из алгоритмов сортировки – пирамидальной сортировкой HeapTree, написали программу для сортировки массивов строк с его использованием, а также провели эксперимент для оценки сложности данного алгоритма. Полученная программа работает исправно и позволяет достаточно быстро сортировать массивы из тысяч строк. Это же подтверждается и результатами эксперимента: алгоритм имеет сложность вида  $O(n * \log(n))$  и, к тому же, требует константное значение дополнительной памяти, т. е. не зависит от размера входных данных.

Всё это говорит о том, что пирамидальная сортировка является одной из самых эффективных для сортировки большого объёма данных, однако алгоритм также имеет и недостатки, например неустойчивость и выигрыш в производительности только на больших значениях  $n$ . Таким образом, в определённых ситуациях целесообразнее использовать другие алгоритмы сортировки, более эффективные для выбранной задачи.

## Литература

1. Левитин А. В. Глава 6. Метод преобразования: Пирамиды и пирамидальная сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 275—284. — 576 с.
2. Федоряева Т. И. Комбинаторные алгоритмы: Учебное пособие /Новосиб. гос. ун-т. Новосибирск, 2011. 118 с.
3. [https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F\\_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) Пирамидальная сортировка
4. <https://habr.com/ru/company/otus/blog/460087/> Пирамидальная сортировка (HeapSort)

## ПРИЛОЖЕНИЕ

### Код класса Program:

```
using System;
using System.Linq;
using System.Xml;

namespace Лаб2_Сортировка_строк
{
    class Program
    {
        static void Main(string[] args)
        {
            if (!Test())
                return;
            Console.WriteLine("Выполнить эксперимент перед началом программы (yes/no)? ");
            var answer = Console.ReadLine();
            if (answer == "yes")
                GetExperimentResults();
            Console.WriteLine("Введите строки для сортировки, разделяя их пробелом: ");
            var input = Console.ReadLine().Split();
            HeapSort(input);
            Console.WriteLine("Отсортированный массив: ");
            PrintArray(input);
        }

        public static int HeapSort(string[] array) // Сортировка массива
        {
            var heapSize = array.Length;
            var swapsCount = 0; // Счётчик swap-ов
            var branchesCount = 0; // Счётчик ветвлений

            for (var i = heapSize / 2 - 1; i >= 0; i--) // Построение кучи
                (перегруппируем массив)
                Heapify(array, heapSize, i, ref swapsCount, ref branchesCount);

            for (var i = heapSize - 1; i >= 0; i--) // По очереди извлекаем элементы из
                кучи
                {
                    var swap = array[0]; // Перемещаем текущий корень в конец
                    array[0] = array[i];
                    array[i] = swap;
                    swapsCount++;
                    Heapify(array, i, 0, ref swapsCount, ref branchesCount); // Вызываем
                    процедуру heapify на уменьшенной куче
                }
            return swapsCount >= branchesCount ? swapsCount : branchesCount; //
            Возвращаем счётчик самой частой операции
        }

        private static void Heapify(string[] array, int heapSize, int i, ref int
            swapsCount, ref int branchesCount) // Преобразования в двоичную кучу размера heapSize
            поддерева с корневым узлом i (индекс в array)
        {
            var largestElem = i; // Инициализируем наибольший элемент как корень
            var leftChild = 2 * i + 1; // left = 2 * i + 1
            var rightChild = 2 * i + 2; // right = 2 * i + 2

```

```

        if (leftChild < heapSize && String.Compare(array[leftChild],
array[largestElem], StringComparison.Ordinal) > 0) // Если левый дочерний элемент больше
корня
            largestElem = leftChild;

        if (rightChild < heapSize && String.Compare(array[rightChild],
array[largestElem], StringComparison.Ordinal) > 0) // Если правый дочерний элемент
больше, чем самый большой элемент на данный момент
            largestElem = rightChild;

        if (largestElem != i) // Если самый большой элемент не корень
        {
            var swap = array[i];
            array[i] = array[largestElem];
            array[largestElem] = swap;
            swapsCount++;
            Heapify(array, heapSize, largestElem, ref swapsCount, ref branchesCount);
// Рекурсивно преобразуем в двоичную кучу затронутое поддерево
        }
        branchesCount += 3;
    }

private static void PrintArray(string[] array) // Вывод массива строк на экран
{
    for (var i = 0; i < array.Length; i++)
        Console.Write(array[i] + " ");
    Console.Read();
}

private static void GetExperimentResults() // Выполнение эксперимента
{
    var xDoc = new XmlDocument();
    xDoc.Load(@"C:\\Users\\Artem\\source\\repos\\Лаб2 Сортировка строк\\Лаб2
Сортировка строк\\Experiment.xml"); // Получаем xml-файл эксперимента
    var xRoot = xDoc.DocumentElement;
    var experiment = xRoot.SelectSingleNode("experiment");
    foreach (XmlNode node in experiment.ChildNodes)
    {
        var minElement = int.Parse(node.SelectSingleNode("@minElement").Value);
// Получение значений атрибутов каждого node
        var maxElement = int.Parse(node.SelectSingleNode("@maxElement").Value);
        var startLength = int.Parse(node.SelectSingleNode("@startLength").Value);
        var maxLength = int.Parse(node.SelectSingleNode("@maxLength").Value);
        var repeat = int.Parse(node.SelectSingleNode("@repeat").Value);
        var name = node.SelectSingleNode("@name").Value;
        if (name == "Arithmetic Progression")
        {
            var diff = int.Parse(node.SelectSingleNode("@diff").Value);
            Console.WriteLine("Эксперимент: " + name);
            for (var length = startLength; length <= maxLength; length += diff)
                Console.WriteLine("Длина массива: " + length + "\\t" + "Кол-во
массивов: " + repeat + "\\t" + "Операций в среднем: " + GetOperationsCount(minElement,
maxElement, repeat, length) / repeat);
        }
        if (name == "Geometric Progression")
        {
            var znamen = double.Parse(node.SelectSingleNode("@znamen").Value);
            Console.WriteLine("Эксперимент: " + name);
            for (var length = startLength; length <= maxLength; length =
(int)Math.Round(length * znamen))
                Console.WriteLine("Длина массива: " + length + "\\t" + "Кол-во
массивов: " + repeat + "\\t" + "Операций в среднем: " + GetOperationsCount(minElement,
maxElement, repeat, length) / repeat);
        }
    }
}

```

```

    }
    Console.WriteLine("\n");
}
}

private static int GetOperationsCount(int minElement, int maxElement, int
repeatsCount, int arrayLength) // Получаем количество операций для массивов заданной
длины
{
    var operationsCount = 0;
    for (var i = 0; i < repeatsCount; i++)
    {
        var array = new string[arrayLength];
        var random = new Random();
        for (var j = 0; j < array.Length; j++)
            array[j] = random.Next(minElement, maxElement).ToString();
        operationsCount += HeapSort(array);
    }
    return operationsCount;
}

private static bool Test() // Тестовая функция
{
    var isAllTestsCompleted = true;
    {
        var startArray = new[] { "bdg", "avc", "hdf", "aa", "dg", "chd" };
        var expectedArray = new[] { "aa", "avc", "bdg", "chd", "dg", "hdf" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 1 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "enfgh", "sgasf", "asdg", "ksd", "asa", "bhj",
"ksgd", "assa", "bhdfj" };
        var expectedArray = new[] { "asa", "asdg", "assa", "bhdfj", "bhj",
"enfgh", "ksd", "ksgd", "sgasf" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 2 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "mbdsdfg", "avghssdc", "hjsdf", "abfga",
"dhjsg", "lnvchd" };
        var expectedArray = new[] { "abfga", "avghssdc", "dhjsg", "hjsdf",
"lnvchd", "mbdsdfg" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 3 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "ajb", "aja", "agah", "agga", "aswgs", "aswgc",
"aba", "aab" };
        var expectedArray = new[] { "aab", "aba", "agah", "agga", "aja", "ajb",
"aswgc", "aswgs" };
    }
}

```

```

        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 4 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    {
        var startArray = new[] { "baggage", "auto", "holiday", "bug", "dog",
"child" };
        var expectedArray = new[] { "auto", "baggage", "bug", "child", "dog",
"holiday" };
        HeapSort(startArray);
        if (!Enumerable.SequenceEqual(startArray, expectedArray))
        {
            Console.WriteLine("Test 5 wasn't passed");
            isAllTestsCompleted = false;
        }
    }
    return isAllTestsCompleted;
}
}
}

```

### Содержимое xml-файла Experiment:

```

<?xml version="1.0" encoding="utf-8" ?>
<experiments>
    <experiment name="HeapTree">
        <nodes name="Arithmetic Progression" minElement="0" maxElement="300"
startLength="1000" diff="100" maxLength="3000" repeat="15">
            </nodes>
        <nodes name="Arithmetic Progression" minElement="0" maxElement="1000"
startLength="2000" diff="100" maxLength="5000" repeat="15">
            </nodes>
        <nodes name="Arithmetic Progression" minElement="0" maxElement="
10000" startLength="1000" diff="100" maxLength="10000" repeat="15">
            </nodes>
        <nodes name="Geometric Progression" minElement="0" maxElement="20"
startLength="10" Znamen="2" maxLength="10000" repeat="15">
            </nodes>
        <nodes name="Geometric Progression" minElement="0" maxElement="1000"
startLength="10" Znamen="5" maxLength="10000" repeat="15">
            </nodes>
    </experiment>
</experiments>

```