МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Новые языки программирования.

Студент гр. 3381 _____ Борисов Е. А.

Студент гр. 3381 _____ Царегородцев Д. И.

Студент гр. 3381 _____ Самигулин Д. А.

Руководитель _____ Шестопалов Р. П.

Санкт-Петербург

2025

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Борисов Е. А. группы 3381

Студент Царегородцев Д. И. группы 3381

Студент Самигулин Д. А. группы 3381

Тема практики: Новые языки программирования

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Kotlin с графическим интерфейсом.

Алгоритм: алгоритм Кнута-Морриса-Пратта.

Сроки прохождения практики: 25.06.2024 – 08.07.2024

Дата сдачи отчета: 02.07.2024

Дата защиты отчета:__.07.2024

Студент гр. 3381	_____	Борисов Е. А.
Студент гр. 3381	_____	Царегородцев Д. И.
Студент гр. 3381	_____	Самигулин Д. А.
Руководитель	_____	Шестопалов Р. П.

АННОТАЦИЯ

Целью учебной практики является изучение и освоение нового языка программирования Kotlin, и последующая реализация алгоритма Кнута-Морриса-Пратта на данном языке программирования. Так же, требуется разработать графический визуализатор алгоритма с графическим интерфейсом.

SUMMARY

The goal of the educational practice is to study and master a new programming language, Kotlin, and subsequently implement the Knuth-Morris-Pratt algorithm in this programming language. It is also required to develop a graphical visualizer of the algorithm with a graphical interface.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является изучение и освоение нового языка программирования Kotlin, и последующая реализация алгоритма Кнута-Морриса-Пратта на данном языке программирования, а также разработка визуализатора алгоритма с графическим интерфейсом. Реализуемый алгоритм используется для поиска подстроки в строке и демонстрирует высокую скорость работы в сравнении с аналогичными алгоритмами, что делает его оптимальным инструментом в задачах поиска совпадений в строке, например, поиска совпадений в цепочках ДНК в задачах биоинформатики. Разработанный графический интерфейс может использоваться в качестве помощника при изучении данного алгоритма, так как наглядно показывает этапы работы алгоритма.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1. Поля ввода: Под шапкой располагаются два текстовых поля: строка A(Text) — исходная строка, в которой ищем. Строка B (Pattern) — шаблон, который ищем в тексте или строим для него префикс-функцию.

2. Вкладки: Ниже полей ввода — две вкладки: 1. КМП — визуализация поиска подстроки или циклического сдвига. 2. Префикс-функция — поэтапная постройка префикс-функции.

2.1 Вкладка «КМП»:

a) Выбор режима: Две кнопки переключают между: классическим поиском подстроки. Циклический сдвиг — проверка, можно ли получить B как циклический сдвиг A. Выбранный режим подсвечивается другим цветом.

b) Инициализация: Кнопка «Запуск»: запускает алгоритм: В режиме классического поиска генерируются все шаги КМП-машины, после чего внизу появляется сообщение с номерами найденных вхождений. В режиме Циклического сдвига строится трасса, а снизу показывается сдвиг (или сообщение «Не найден циклический сдвиг»).

c) Клавиши управления: Реализация стандартного плеера (шаг назад, старт/пауза алгоритма, шаг вперед, сразу к последнему шагу, слайдер скорости воспроизведения).

2.2 Вкладка «Префикс-функция»:

a) Постройка префикс-функции: Запуск поэтапного вычисления префикс-функции для строки B.

b) Управление анимацией: Те же кнопки и слайдер скорости, что и во вкладке поиска, позволяют листать шаги алгоритма построения префикс-функции.

Основные принципы работы:

1. Принцип шаг за шагом: алгоритм сначала генерирует список шагов (каждый шаг содержит индексы в A и B и текущее значение префикс-функции), а потом мы просто пролистываем этот список.

2. Анимация: при включённом проигрывании автоматически переходим на следующий шаг через указанный интервал.

3. Интерактивность: можно вручную перемещаться по шагам и менять скорость.

4. Обратная связь: по завершении работы появляется всплывающее окно с результатом поиска или сдвига.

1.2. Уточнения после сдачи прототипа

Необходимо добавить возможность считывания входных данных из файла, изменить отображение совпадений, а именно подсвечивать совпадения на своих позициях. Добавить отдельное поле с результатом работы и сохранение результата в файл.

1.3 Уточнения после сдачи первой версии

Необходимо добавить горизонтальную прокрутку для последовательностей большого размера, а также автопрокрутку с центрированием отображения на текущем шаге алгоритма.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
30.06.2025	Согласование спецификации	Написание требований, плана тестирования, реализация визуализации и алгоритма	Выполнено
02.07.2025	Сдача прототипа	Реализован графический интерфейс. Добавлены возможности выбора между обычным КМП и нахождением циклического сдвига, изменение скорости отображения, визуализация построения префикс-функции.	Выполнено
04.07.2025	Сдача версии 1	Добавлена возможность считывания данных из файла, исправлено отображение найденных совпадений, добавлено отдельное поле с результатом работы и сохранением в файл.	Выполнено
07.07.2025	Сдача версии 2	Добавлена горизонтальная	

		прокрутка для последовательностей большого размера. Добавлена автопрокрутка больших последовательностей с центрированием отображения	
07.07.2025	Сдача отчёта		
07.07.2025	Защита отчёта		

2.2. Распределение ролей в бригаде

Студент Борисов Е. А. группы 3381 – тестирование программы на этапах разработки, составление отчёта о выполненной работе.

Студент Царегородцев Д. И. группы 3381 – реализация алгоритма Кнута-Морриса-Пратта, составление плана разработки и спецификации.

Студент Самигулин Д. А. группы 3381 – реализация визуализатора алгоритма с графическим интерфейсом.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Основные структуры Kmp.kt

KmpStep(val i: Int, val j: Int, val pi, List<Int>): класс для хранения шагов работы алгоритма. Содержит в себе следующие параметры: i – текущая позиция в тексте, j – текущая позиция в шаблоне, pi – текущая префикс-функция.

PiStep(val i: Int, val j: Int, val pi, List<Int>): класс для хранения шагов построения префикс-функции шаблона. Содержит в себе следующие параметры: i – текущий индекс в образце, j – текущая длина совпадающих префикса и суффикса, pi – текущее состояние префикс-функции.

3.2. Основные методы Kmp.kt

buildPi(p: String): List<Int>: функция вычисляет префикс-функцию для переданной строки. Возвращает список целых чисел, являющийся префикс-функцией.

generateSteps(text: String, pattern: String): List<KmpStep>: функция является реализацией алгоритма Кнута-Морриса-Пратта. Осуществляет поиск совпадений шаблона в тексте в соответствии с вышеуказанным алгоритмом и возвращает список шагов алгоритма в формате KmpStep.

piSteps(p: String): List<PiStep>: функция является реализацией пошагового построения префикс-функции. Возвращает список шагов построения в формате PiStep.

cyclicShift(A: String, B: String): Pair<Int, List<KmpStep>>: функция реализует проверку текста на циклический сдвиг шаблона.

3.3 Основные методы Main.kt

CharBox(ch: Char, bg: Color): функция для отображения символа внутри квадрата. Создаёт квадратную область, сглаживает углы области, задаёт цвет и отображает символ по центру области.

ScrollTape(text: String, pos: Int, ranges: List<IntRange>, playing: Boolean, listState: LazyListState): функция для отображения горизонтально прокручиваемой ленты для возможности корректного показа последовательностей больших размеров. При анимации смещения позиции на этапах выполнения алгоритма лента прокручивается автоматически, чтобы текущее место выполнения не выпадало из поля отображения. Так же доступна прокрутка вручную.

Tape(text: String, pos:Int): функция, аналогичная предыдущей, но для последовательностей, которым не требуется горизонтальная прокрутка. Подсвечивает текущую позицию.

App(): функция, реализующая полный графический интерфейс для взаимодействия с конечным пользователем. В начале функции инициализируются состояния и вспомогательные объекты. Создаются диалоговые окна для работы с файлами. Основные строки text и pat хранят входные данные для алгоритма, с начальными значениями "defabc" и "abcdef" соответственно. Переменная mode определяет режим работы (поиск подстроки SUBSTRING или циклического сдвига CYCLIC), a tab управляет активной вкладкой интерфейса (KMP run или π-function).

Для визуализации основного алгоритма используются несколько состояний: steps хранит список шагов алгоритма, idx - текущий шаг, playing - флаг воспроизведения анимации, speed - скорость анимации. Аналогичные состояния piSteps, piIdx, piPlaying и piSpeed используются для визуализации префикс-функции. Эффекты LaunchedEffect управляют автоматическим воспроизведением анимации.

Основное содержимое размещается в Box. Первые элементы интерфейса - это поля ввода текста и шаблона с кнопками загрузки из файла. При нажатии на кнопку открывается диалог выбора файла, содержимое которого загружается в соответствующее поле. Ниже расположен переключатель режимов работы. Для каждого значения перечисления Mode создается кнопка Button, цвет которой зависит от текущего выбранного режима.

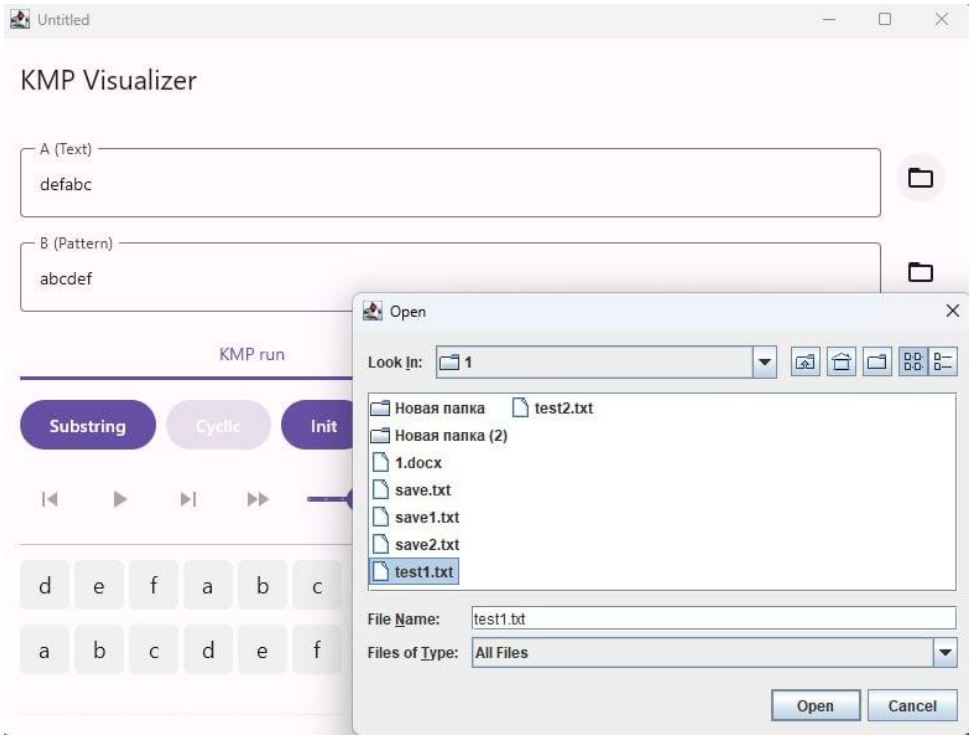
Управление анимацией KMP включает кнопку "Init", которая инициализирует steps в зависимости от выбранного режима. Ряд кнопок позволяет управлять воспроизведением: переключатель play/pause, шаг назад/вперед, полная перемотка в конец, ползунок для регулировки скорости анимации. Текущие позиции в тексте и шаблоне подсвечиваются разными цветами: синим для текущей позиции, зелёным для найденных совпадений.

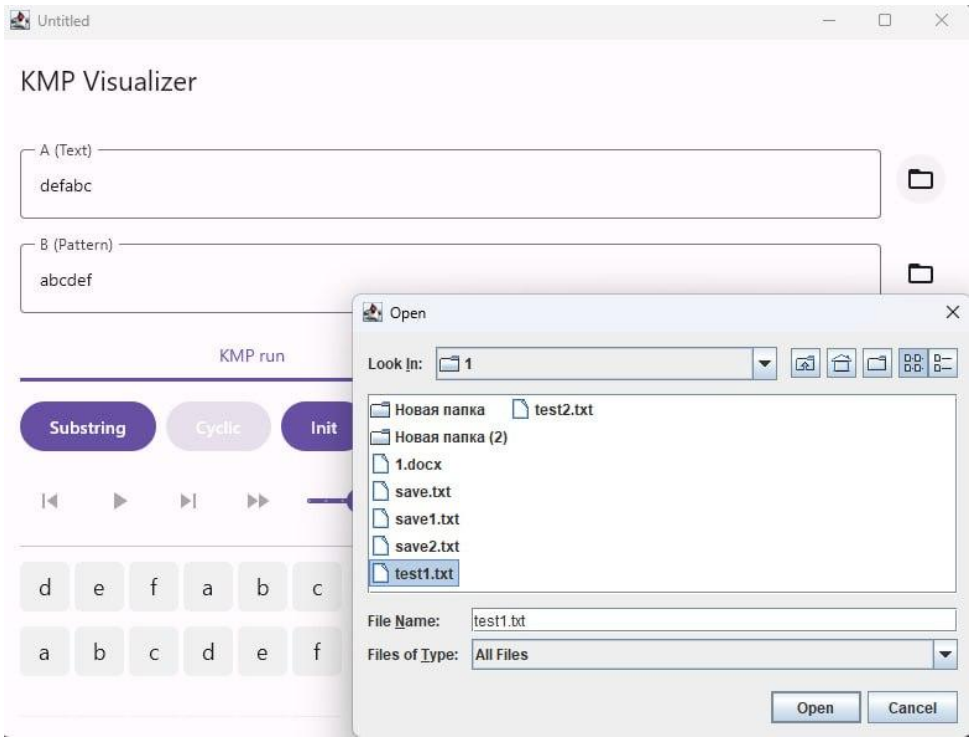
Результаты работы алгоритма отображаются в нижнем правом углу. В зависимости от режима формируется сообщение message - либо список найденных позиций, либо список циклических сдвигов. Рядом находится кнопка сохранения, которая открывает диалог выбора файла и сохраняет туда отчет с входными данными, результатами и значениями префикс-функции.

4. ТЕСТИРОВАНИЕ

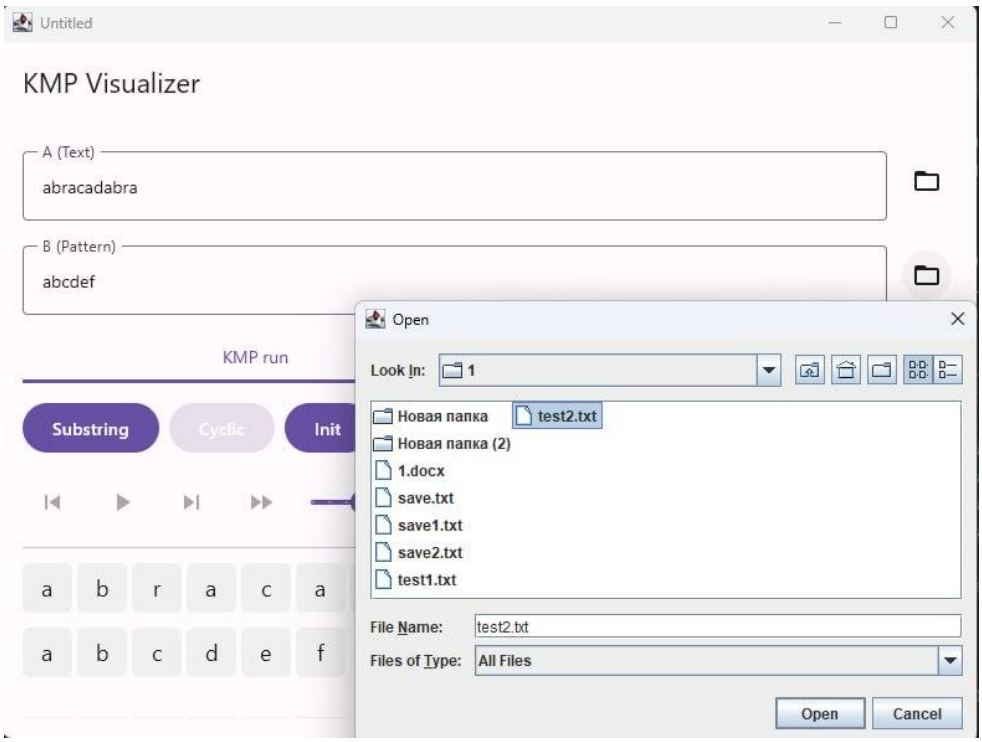
4.1. Тестирование графического интерфейса

1) Считывание данных из файла

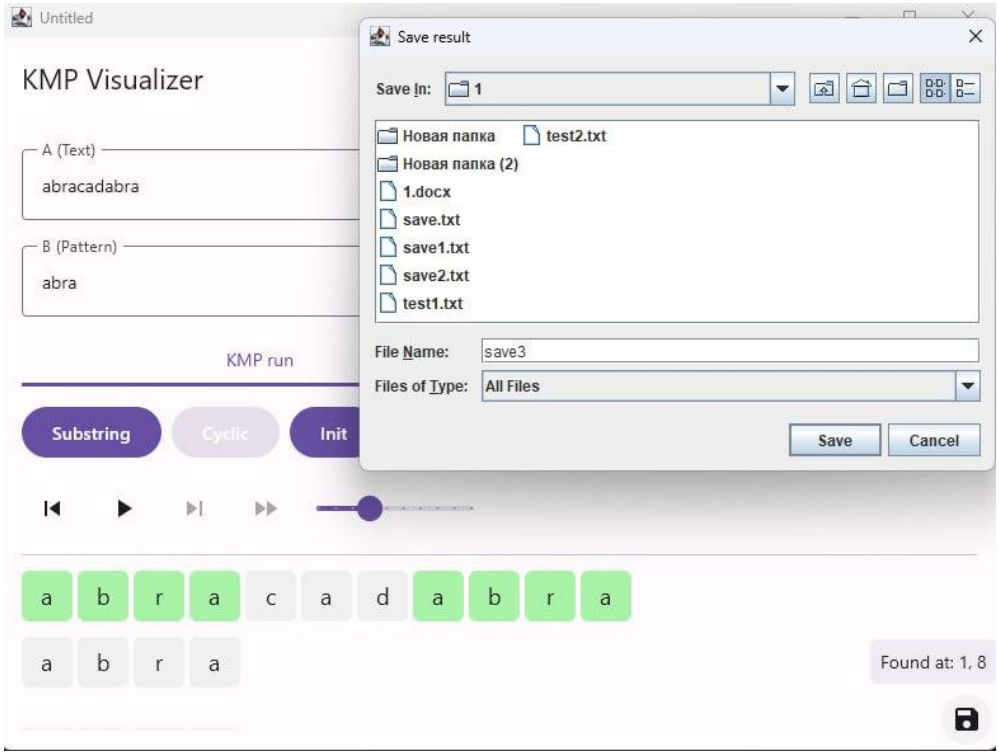


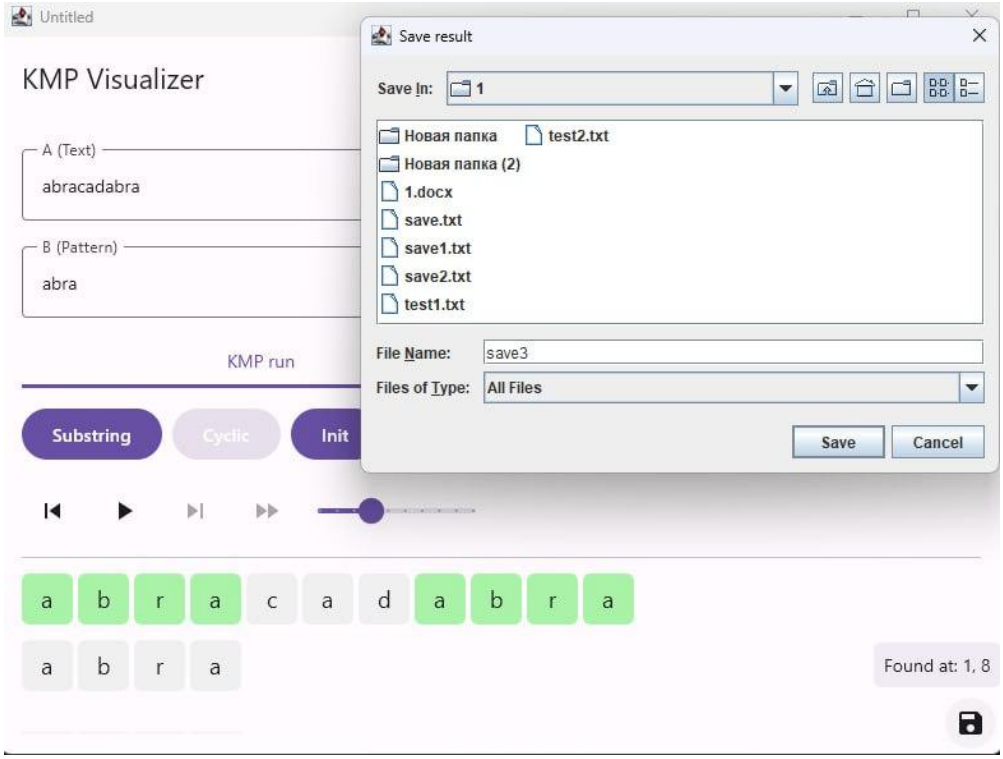


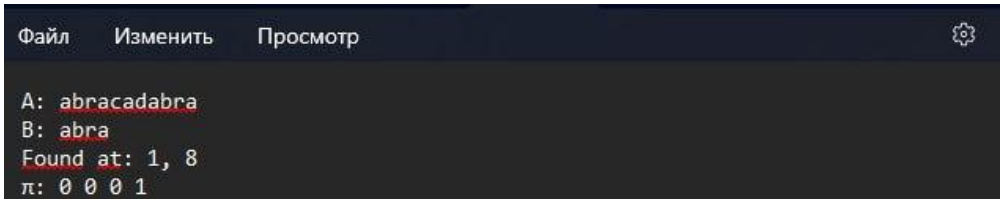
2) Считывание данных с клавиатуры



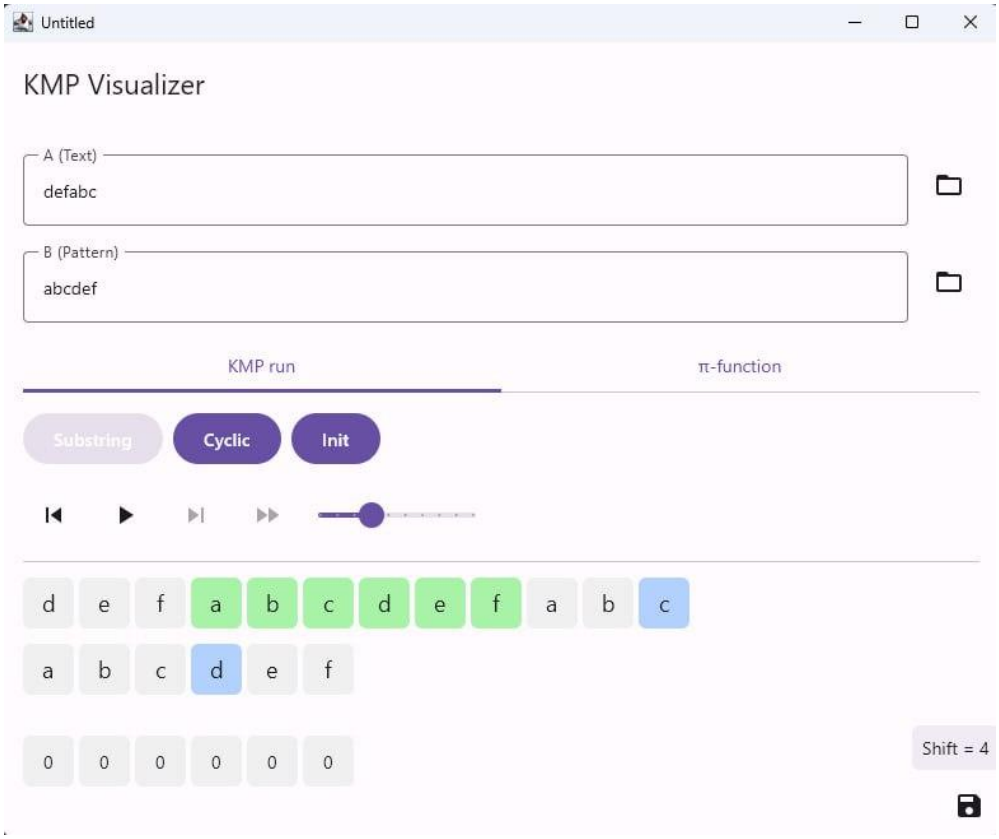
3) Отображение результатов в окне и сохранение в файл







4) Результат работы проверки циклического сдвига



ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был изучен язык программирования Kotlin. Был разработан визуализатор алгоритма Кнута-Морриса-Пратта с графическим интерфейсом. Разработанный визуализатор наглядно демонстрирует принцип работы данного алгоритма, а также принцип построения префикс-функции, и может использоваться в качестве помощника при изучении данного алгоритма. Конечный результат работы соответствует заявленным требованиям, а также уточнениям в процессе сдачи программы на разных этапах разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Онлайн-курс Kotlin на платформе Stepik. URL: https://stepik.org/course/5448/syllabus

2. Викиконспекты ИТМО. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BD%D1%83%D1%82%D0%B0-%D0%9C%D0%BE%D1%80%D1%80%D0%B8%D1%81%D0%B0-%D0%9F%D1%80%D0%B0%D1%82%D1%82%D0%B0

3. Официальная документация Kotlin. URL: https://kotlinlang.org/docs/home.html

4. Metanit.com. Руководство по языку Kotlin. URL: https://metanit.com/kotlin/tutorial/?ysclid=mcomoel833792378774

5. developer.android.com. Руководство по составлению Jetpack. URL: https://developer.android.com/develop/ui/compose/tutorial?hl=ru

ПРИЛОЖЕНИЕ А
НАЗВАНИЕ ПРИЛОЖЕНИЯ

Файл Kmp.kt

```
package org.example.algorithm

/** ---------- шаги для визуализации ---------- */
data class KmpStep(val i: Int, val j: Int, val pi: List<Int> = emptyList())
data class PiStep (val i: Int, val j: Int, val pi: List<Int>)

/** ---------- класс-обёртка над KMP ---------- */
class Kmp {

    /** префикс-функция */
    private fun buildPi(p: String): List<Int> {
        val pi = MutableList(p.length) { 0 }
        var j = 0
        for (i in 1 until p.length) {
            while (j > 0 && p[i] != p[j]) j = pi[j - 1]
            if (p[i] == p[j]) j++
            pi[i] = j
        }
        return pi
    }

    /** пошаговый KMP-поиск `pattern` в `text` */
    fun generateSteps(text: String, pattern: String): List<KmpStep> {
        val out = mutableListOf<KmpStep>()
        if (pattern.isEmpty()) return out
        val pi = buildPi(pattern)
        var j = 0
        for (i in text.indices) {
            while (j > 0 && text[i] != pattern[j]) {
                out += KmpStep(i, j, pi); j = pi[j - 1]
            }
            if (text[i] == pattern[j]) {
                out += KmpStep(i, j, pi)     // до ++
                j++
                out += KmpStep(i, j, pi)     // после ++
                if (j == pattern.length) j = pi[j - 1]
            } else out += KmpStep(i, j, pi)
        }
        return out
    }

    /** пошаговая постройка π-массива */
    fun piSteps(p: String): List<PiStep> {
        val pi = MutableList(p.length) { 0 }
        val out = mutableListOf<PiStep>()
        var j = 0
        for (i in 1 until p.length) {
            while (j > 0 && p[i] != p[j]) { j = pi[j - 1]; out += PiStep(i, j, pi.toList()) }
            if (p[i] == p[j]) j++
            pi[i] = j
            out += PiStep(i, j, pi.toList())
        }
        return out
    }
```

```
    /** проверка циклического сдвига.
     *  Возвращает (индекс начала `pattern` в `text`, trace-шаги) */
    fun cyclicShift(A: String, B: String): Pair<Int, List<KmpStep>> {
        if (A.length != B.length) return -1 to emptyList()
        if (A.isEmpty())          return 0  to emptyList()

        val text  = A + A                    // «раскручиваем» строку
        val trace = generateSteps(text, B)
        val matchEnd = trace
            .firstOrNull { it.j == B.length } // первый шаг, где j == |B|
            ?.i                               // позиция последнего совпавшего символа

        val shift = matchEnd
            ?.let { it - B.length + 1 }       // переводим в начало
            ?.takeIf { it < A.length } ?: -1  // валиден только < |A|

        return shift to trace
    }
}
```

Файл Main.kt

```
package org.example

import androidx.compose.animation.animateColorAsState
import androidx.compose.foundation.HorizontalScrollbar
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyListState
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.lazy.rememberLazyListState
import androidx.compose.foundation.rememberScrollbarAdapter
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application
import kotlinx.coroutines.delay
import kotlinx.coroutines.isActive
import org.example.algorithm.*
import java.io.File
import javax.swing.JFileChooser

private val CELL: Dp = 40.dp
private val SHAPE = RoundedCornerShape(6.dp)
private val C_BASE = Color(0xFFF0F0F0)
private val C_CURR = Color(0xFFB3D1FF)
private val C_MATCH = Color(0xFFA8F3A8)

@Composable
private fun CharBox(ch: Char, bg: Color) =
```

```
    Surface(Modifier.size(CELL), SHAPE, bg, tonalElevation = 4.dp) {
        Box(Modifier.fillMaxSize(), Alignment.Center) { Text(ch.toString(), fontSize = 18.sp) }
    }

@Composable
private fun ScrollTape(
    text: String,
    pos: Int,
    ranges: List<IntRange>,
    playing: Boolean,
    listState: LazyListState
) {
    val cellPx = with(LocalDensity.current) { (CELL + 4.dp).roundToPx() }
    LaunchedEffect(pos, playing) {
        if (pos < 0) return@LaunchedEffect
        val info = listState.layoutInfo
        val vp = info.viewportSize.width
        if (playing) {
            val first = (pos - vp / (2 * cellPx)).coerceAtLeast(0)
            listState.animateScrollToItem(first)
        } else {
            val last = info.visibleItemsInfo.lastOrNull()?.index ?: 0
            val first = info.visibleItemsInfo.firstOrNull()?.index ?: 0
            when {
                pos >= last - 1 -> listState.animateScrollToItem(first + 3)
                pos <= first + 1 -> listState.animateScrollToItem((pos - 3).coerceAtLeast(0))
            }
        }
    }

    Box {
        LazyRow(
            state = listState,
            horizontalArrangement = Arrangement.spacedBy(4.dp),
            modifier = Modifier
                .fillMaxWidth()
                .height(CELL)
        ) {
            itemsIndexed(text.toList()) { i, c ->
                val bg by animateColorAsState(
                    when {
                        ranges.any { i in it } -> C_MATCH
                        i == pos && pos >= 0 -> C_CURR
                        else -> C_BASE
                    }, label = ""
                )
                CharBox(c, bg)
            }
        }
        HorizontalScrollbar(
            rememberScrollbarAdapter(listState),
            Modifier
                .align(Alignment.BottomStart)
                .fillMaxWidth()
        )
    }
}

@Composable
private fun Tape(text: String, pos: Int) =
    Row(horizontalArrangement = Arrangement.spacedBy(4.dp)) {
        text.forEachIndexed { i, c ->
```

```
            val bg by animateColorAsState(if (i == pos && pos >= 0) C_CURR else C_BASE, label = "")
            CharBox(c, bg)
        }
    }

private enum class Mode { SUBSTRING, CYCLIC }

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun App() {
    val chooserA = remember { JFileChooser() }
    val chooserB = remember { JFileChooser() }
    val saveDialog = remember { JFileChooser().apply { dialogTitle = "Save result" } }

    var text by remember { mutableStateOf("defabc") }
    var pat by remember { mutableStateOf("abcdef") }

    var mode by remember { mutableStateOf(Mode.SUBSTRING) }
    var tab by remember { mutableStateOf(0) }
    val tabs = listOf("KMP run", "π-function")

    val kmp = remember { Kmp() }

    var steps by remember { mutableStateOf(emptyList<KmpStep>()) }
    var idx by remember { mutableStateOf(0) }
    var playing by remember { mutableStateOf(false) }
    var speed by remember { mutableFloatStateOf(700f) }

    var piSteps by remember { mutableStateOf(emptyList<PiStep>()) }
    var piIdx by remember { mutableStateOf(0) }
    var piPlaying by remember { mutableStateOf(false) }
    var piSpeed by remember { mutableFloatStateOf(500f) }

    LaunchedEffect(playing, idx, steps, speed) {
        while (playing && isActive) {
            delay(speed.toLong())
            if (idx < steps.lastIndex) idx++ else playing = false
        }
    }
    LaunchedEffect(piPlaying, piIdx, piSteps, piSpeed) {
        while (piPlaying && isActive) {
            delay(piSpeed.toLong())
            if (piIdx < piSteps.lastIndex) piIdx++ else piPlaying = false
        }
    }

    val listState = rememberLazyListState()

    Scaffold(topBar = { TopAppBar(title = { Text("KMP Visualizer") }) }) { pad ->
        Box(Modifier.fillMaxSize()) {
            Column(
                Modifier
                    .padding(pad)
                    .padding(16.dp)
                    .fillMaxWidth(),
                verticalArrangement = Arrangement.spacedBy(12.dp)
            ) {
                Row(Modifier.fillMaxWidth(), Arrangement.spacedBy(8.dp), Alignment.CenterVertically) {
                    OutlinedTextField(text, { text = it }, Modifier.weight(1f), label = { Text("A") })
                    IconButton({
                        if (chooserA.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
                            text = chooserA.selectedFile.readText()
```

```
            }) { Icon(Icons.Default.FolderOpen, null) }
        }
        Row(Modifier.fillMaxWidth(), Arrangement.spacedBy(8.dp), Alignment.CenterVertically) {
            OutlinedTextField(pat, { pat = it }, Modifier.weight(1f), label = { Text("B") })
            IconButton({
                if (chooserB.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
                    pat = chooserB.selectedFile.readText()
            }) { Icon(Icons.Default.FolderOpen, null) }
        }

        TabRow(selectedTabIndex = tab) {
            tabs.forEachIndexed { i, t -> Tab(tab == i, { tab = i }) { Text(t, Modifier.padding(12.dp)) } }
        }

        if (tab == 0) {
            Row(horizontalArrangement = Arrangement.spacedBy(8.dp), verticalAlignment =
Alignment.CenterVertically) {
                Mode.values().forEach { m ->
                    val sel = mode == m
                    Button(
                        { mode = m },
                        colors = ButtonDefaults.buttonColors(
                            if (sel) MaterialTheme.colorScheme.primary
                            else MaterialTheme.colorScheme.surfaceVariant
                        )
                    ) { Text(m.name.lowercase().replaceFirstChar { it.titlecase() }) }
                }
                Button({
                    steps = if (mode == Mode.SUBSTRING) kmp.generateSteps(text, pat)
                    else kmp.cyclicShift(text, pat).second
                    idx = 0; playing = false
                }) { Text("Init") }
            }

            Row(horizontalArrangement = Arrangement.spacedBy(8.dp), verticalAlignment =
Alignment.CenterVertically) {
                IconButton({ idx-- }, enabled = idx > 0) { Icon(Icons.Default.SkipPrevious, null) }
                IconToggleButton(playing, { playing = it }, enabled = steps.isNotEmpty()) {
                    Icon(if (playing) Icons.Default.Pause else Icons.Default.PlayArrow, null)
                }
                IconButton({ idx++ }, enabled = idx < steps.lastIndex) { Icon(Icons.Default.SkipNext, null) }
                IconButton({ idx = steps.lastIndex }, enabled = idx < steps.lastIndex) {
                    Icon(Icons.Default.FastForward, null)
                }
                Slider(value = speed, onValueChange = { speed = it }, valueRange = 200f..1500f,
                    steps = 8, modifier = Modifier.width(140.dp))
            }

            Divider()

            val step = steps.getOrNull(idx)
            val ranges = steps.take(idx + 1).filter { it.j == pat.length }
                .map { it.i - pat.length + 1..it.i }

            val full = if (mode == Mode.SUBSTRING) text else text + text
            ScrollTape(full, step?.i ?: -1, ranges, playing, listState)
            Tape(pat, step?.j ?: -1)

            Spacer(Modifier.height(8.dp))
            Row(horizontalArrangement = Arrangement.spacedBy(4.dp)) {
                (step?.pi ?: List(pat.length) { 0 }).forEach {
                    Surface(Modifier.size(CELL), SHAPE, C_BASE, tonalElevation = 2.dp) {
```

```
                    Box(Modifier.fillMaxSize(), Alignment.Center) { Text(it.toString()) }
                }
            }
        }
    } else {
        Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
            Button({
                piSteps = kmp.piSteps(pat); piIdx = 0; piPlaying = false
            }) { Text("Build π") }
            IconButton({ piIdx-- }, enabled = piIdx > 0) { Icon(Icons.Default.SkipPrevious, null) }
            IconToggleButton(piPlaying, { piPlaying = it }, enabled = piSteps.isNotEmpty()) {
                Icon(if (piPlaying) Icons.Default.Pause else Icons.Default.PlayArrow, null)
            }
            IconButton({ piIdx++ }, enabled = piIdx < piSteps.lastIndex) { Icon(Icons.Default.SkipNext, null) }
            IconButton({ piIdx = piSteps.lastIndex }, enabled = piIdx < piSteps.lastIndex) {
                Icon(Icons.Default.FastForward, null)
            }
            Slider(value = piSpeed, onValueChange = { piSpeed = it }, valueRange = 200f..1500f,
                steps = 8, modifier = Modifier.width(140.dp))
        }

        Divider()

        val p = piSteps.getOrNull(piIdx)
        Tape(pat, p?.i ?: -1)
        Spacer(Modifier.height(8.dp))
        Row(horizontalArrangement = Arrangement.spacedBy(4.dp)) {
            (p?.pi ?: List(pat.length) { 0 }).forEach {
                Surface(Modifier.size(CELL), SHAPE, C_BASE, tonalElevation = 2.dp) {
                    Box(Modifier.fillMaxSize(), Alignment.Center) { Text(it.toString()) }
                }
            }
        }
    }
}

val found = steps.take(idx + 1).filter { it.j == pat.length }
    .map { it.i - pat.length + 1 }
val message = if (mode == Mode.SUBSTRING) {
    if (found.isEmpty()) "" else "Found at: ${found.map { it + 1 }.joinToString()}"
} else {
    if (found.isEmpty()) "" else "Shift: ${found.map { (it % pat.length) + 1 }.joinToString()}"
}

Box(Modifier.fillMaxSize(), Alignment.BottomEnd) {
    if (message.isNotBlank()) {
        Column(horizontalAlignment = Alignment.End) {
            Surface(shape = SHAPE, tonalElevation = 4.dp) { Text(message, Modifier.padding(8.dp)) }
            Spacer(Modifier.height(4.dp))
            IconButton({
                if (saveDialog.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
                    val file = saveDialog.selectedFile.let {
                        if (it.extension.isBlank()) File(it.parentFile, "${it.name}.txt") else it
                    }
                    val pi = kmp.piSteps(pat).last().pi
                    val report = buildString {
                        appendLine("A: $text")
                        appendLine("B: $pat")
                        appendLine(message)
                        appendLine("π: ${pi.joinToString(" ")}")
                    }
                    file.writeText(report)
```

```
                    }
                }) { Icon(Icons.Default.Save, null) }
            }
        }
    }
}

fun main() = application { Window(onCloseRequest = ::exitApplication) { App() } }
```