МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ
(КАФЕДРА №43)

ОТЧЁТ
ЗАЩИЩЁН С ОЦЕНКОЙ

ассистент		Д. А. Смолиенко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

отчёт по лабораторной работе №8

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНОГО ПРОДУКТА С ПОМОЩЬЮ ПРОФАЙЛЕРА

по дисциплине: УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

вариант № 4

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР.

4631		Д. С. Воробьев
№ группы	подпись, дата	инициалы, фамилия

Санкт-Петербург

2018 г.

1. Цель работы

Произвести функциональное тестирование кода, оценить его покрытие и качество тестов. Вариант задания:

> Функция поиска пути в неориентированном графе методом A^*. На вход подается карта (граф с географическими координатами вершин) и два угла. На выходе – путь между этими узлами.

10. Взять задание из лабораторной работы номер 2. Модифицировать полученный код чтобы разработанную функцию можно было выполнять большое количество раз (например, 10000).

11. Подключиться к исполняемому коду профилировщиком (или использовать встроенный в IDE) и проанализировать – какой фрагмент кода занял больше всего процессорного времени.

12. Предложить способы оптимизации

2. Текст программы

Листинг 1 — Main.kt

```kotlin
import org.assertj.swing.fixture.FrameFixture
import org.openqa.selenium.chrome.ChromeDriver
import java.awt.Dimension
import java.io.File
import java.util.*

val debug = true

fun main(args: Array<String>) {

    fun maybePrint(any: Any) {
        if (debug)
            println(any)
    }

    fun maybeError(any: Any) {
        if (debug)
            System.err.println(any)
    }

    for (i in 0..(if (debug) 1 else 10000))
        Tests.javaClass.declaredMethods.forEach {
            if (it.isAnnotationPresent(Test::class.java)) {
                try {
                    val tests = it.invoke(Tests)
                    if (it.returnType.canonicalName == "void" ||
                            (it.returnType.canonicalName == "boolean" &&
                                    tests as Boolean))
                        maybePrint("${it.name} passed")
                    else
                        maybeError("${it.name} error")
                } catch (e: Exception) {
                    maybeError("${it.name} error")
                    if (debug)
                        e.printStackTrace()
                }
            }
        }
}

infix fun String.shouldMatch(regex: Regex) {}

infix fun <T, V> T.shouldBe(other: V) {
    if (this != other)
        throw TestException("$this doesn't equal $other")
}

class TestException(msg: String) : RuntimeException(msg)

@Target(AnnotationTarget.FUNCTION)
annotation class Test

object Tests {
    fun loadWayAsString(name: String, from: String, to: String): String {
        val graph = loadGraph("$name.ggb")
        val way = aStar(graph[from], graph[to])
        return way.toString()
    }

    @Test
    fun testLongWay2(): Boolean {
        return loadWayAsString("Graph4", "A", "Z") ==
        ↪  "[A, B, S, T, R, E, M, N, Q, P, Z]"
    }

    @Test
            /** Test:
             * if (current == to) -> true
             * if (last == to) -> true
             */
    fun testB1() {
        val graph = Graph.build {
            val a = it.vertex("a", 1, 1)
        }
        aStar(graph["a"], graph["a"]) shouldBe listOf(graph["a"])
    }

    @Test
            /** Test:
             * if (current == to) -> false
             * if (!handled[it.i] || newCost < costSoFar[it.i]) -> true
             * if (current == to) -> true
             * if (last == to) -> true
             */
    fun testB2() {
        val graph = Graph.build {
            val a = it.vertex("a", 1, 1)
            val b = it.vertex("b", -1, -1)

            a - b
        }
        aStar(graph["a"], graph["b"]) shouldBe listOf(graph["a"], graph["b"])
    }

    @Test
```

```
            /** Test:
             * if (current == to) -> false
             * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
             * if (!handled[it.i] || newCost < costSoFar[it.i]) -> true
             * if (current == to) -> true
             * if (last == to) -> true
             */
    fun testB3() {
        val graph = Graph.build {
            val a = it.vertex("a", 1, 1)
            val b = it.vertex("b", -1, -1)
            val c = it.vertex("c", -2, -2)

            a - b
            a - c
        }
        aStar(graph["a"], graph["c"]) shouldBe listOf(graph["a"], graph["c"])
    }

    @Test
            /** Test:
             * if (current == to) -> false
             * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
             * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
             * while (frontier.isNotEmpty()) -> false
             * if (last == to) -> false
             */
    fun testB4() {
        val graph = Graph.build {
            val a = it.vertex("a", 1, 1)
            val b = it.vertex("b", -1, -1)
            val c = it.vertex("c", -2, -2)

            a - b
        }
        aStar(graph["a"], graph["c"]) shouldBe listOf<Graph.Vertex>()
    }

    @Test
    fun testA1() {
        val graph = Graph.build {
            val a = it.vertex("a", 0, 1)
            val b = it.vertex("b", 1, 0)
            val c = it.vertex("c", 0, 0)
            val d = it.vertex("d", 0, -1)

            a - b
            a - c

            d - b
            d - c
        }
        aStar(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>(graph["a"], graph[
        ↪  "c"], graph["d"])
    }

    @Test
    fun testA2() {
```

```kotlin
            val graph = Graph.build {
                val a = it.vertex("a", 0, 1)
                val b = it.vertex("b", 0, 0)
                val c = it.vertex("c", -1, 0)
                val d = it.vertex("d", 0, -1)

                a - b
                a - c

                d - b
                d - c
            }
            aStar(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>(graph["a"], graph[
            ↪  "b"], graph["d"])
        }

        //@Test
        fun testC1() {
            if (!debug)
                return

            val max = 40
            val rn = Random()
            val sb = StringBuilder()
            for (i in 1..max) {
                for (j in 1..max)
                    sb.appendln("val${i}n$j($i, $j)")
                if (i > 1)
                    for (j in 1..max) {
                        for (k in 1..max)
                            if (rn.nextInt(2) == 0)
                                sb.appendln("val${i - 1}n$j - val${i}n$k")
                    }
                for (j in 1..max) {
                    for (k in 1..max)
                        if (rn.nextInt(2) == 0)
                            sb.appendln("val${i}n$j - val${i}n$k")
                }
            }
            for (i in 1..max)
                sb.appendln("val1n$i ? val${max}n${max + 1 - i}")
            val g = graphFromString(sb.toString())
        }

        @Test
        fun windowTest() {
            if (!debug)
                return

            fun wait(ms: Int) {
                Thread.sleep(ms.toLong())
            }

            val fixure = FrameFixture(Window)
            for (i in 0..4) {
                when (i) {
                    1 -> fixure.resizeTo(Dimension(1000, 200))
                    2 -> fixure.resizeTo(Dimension(200, 1000))
                    3 -> fixure.resizeTo(Dimension(200, 200))
                    4 -> fixure.resizeTo(Dimension(100, 100))
                }

                if (i > 0) {
                    fixure.textBox("inputTextArea").setText("A(1,1)\nA?A")
                    fixure.button("convertButton").click()
                    fixure.textBox("outputTextArea").requireText("[A]")
                    fixure.button("undoButton").click()
                    continue
                }

                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").enterText("A(1,1)\nA?A")
                fixure.button("convertButton").click()
                fixure.textBox("outputTextArea").requireText("[A]")
                ///
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nB(2,2)\nA?B")
                fixure.button("convertButton").click()
                fixure.textBox("outputTextArea").requireText("[]")
                ///
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nB(2,2)\nA-B\nA?B")
                fixure.button("convertButton").click()
                fixure.textBox("outputTextArea").requireText(Regex("\\[A,.*").toPattern())
                ///
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nA-B")
                fixure.button("convertButton").click()
                wait(500)
                fixure.textBox("outputTextArea").requireText(
                        "VertexNotDeclaredException: Vertex \"B\" isn't declared!\n" +

                                ↪   "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                ↪   +

                                ↪   "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                ↪   +

                                ↪   "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
                                ↪   +
                                "\tat Graph\$Companion.build(Graph.kt:65)\n" +
                                "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
                                ↪   +
                                "\tat Step.<init>(Window.kt:22)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:35)\n" +

                                ↪   "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
                )
                ///
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nA(1,1)")
                fixure.button("convertButton").click()
                wait(500)
                fixure.textBox("outputTextArea").requireText(
                        "VertexAlreadyDeclaredException: Vertex \"A\" already declared!\n"
                        ↪  +

                                ↪  "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                ↪  +

                                ↪  "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                ↪  +

                                ↪  "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
                                ↪  +
                                "\tat Graph\$Companion.build(Graph.kt:65)\n" +
                                "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
                                ↪  +
                                "\tat Step.<init>(Window.kt:22)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:35)\n" +

                                 ↪  "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
                )
                ///
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A*B")
                fixure.button("convertButton").click()
                wait(500)
                fixure.textBox("outputTextArea").requireText(
                        "GraphParserException: Line \"A*B\" not recognized.\n" +

                                 ↪  "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                 ↪  +

                                 ↪  "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGrap
                                 ↪  +

                                 ↪  "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
                                 ↪  +
                                "\tat Graph\$Companion.build(Graph.kt:65)\n" +
                                "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
                                ↪  +
                                "\tat Step.<init>(Window.kt:22)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
                                "\tat Window\$2\$2.invoke(Window.kt:35)\n" +

                                 ↪  "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
                )
                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nB(1,2)\nA-B\nA?B")
                fixure.button("convertButton").click()
                fixure.textBox("outputTextArea").requireText("[A, B]")

                fixure.textBox("inputTextArea").setText("")
                fixure.textBox("inputTextArea").setText("A(1,1)\nB(1,2)\nA-B\nB?A")
                fixure.button("convertButton").click()
                fixure.textBox("outputTextArea").requireText("[B, A]")

                fixure.button("undoButton").click()
                fixure.textBox("inputTextArea").requireText("A(1,1)\nB(1,2)\nA-B\nA?B")
                fixure.textBox("outputTextArea").requireText("[A, B]")
```

```

            for (j in 0..20) {
                fixure.button("undoButton").click()
                wait(50)
            }
        }

        Window.isVisible = false
        /*
        fixure.textBox("outputTextArea").requireText(
                Regex("GraphParser.+").toPattern()
        )
        */
    }

    @Test
    fun webTest() {
        if (!debug)
            return

        WebTest

        fun wait(ms: Int) {
            Thread.sleep(ms.toLong())
        }

        val webDriver = ChromeDriver()
        System.setProperty("webdriver.chrome.driver", File("chromedriver").
        ↪  absolutePath);
        webDriver.navigate().to("http://localhost:8000/test?input=")
        wait(200)

        var input = webDriver.findElementByName("input")
        var submit = webDriver.findElementByName("submitButton")
        input.sendKeys("A(1,1)\nA?A")
        wait(200)
        submit.click()
        wait(200)

        var output = webDriver.findElementByName("output")
        output.text.trim() shouldBe "[A]"
        wait(500)
        input = webDriver.findElementByName("input")
        submit = webDriver.findElementByName("submitButton")
        input.sendKeys("\nB(1,1)\nA?B")
        wait(200)
        submit.click()

        wait(200)
        output = webDriver.findElementByName("output")
        output.text.trim() shouldBe "[A]\n[]"
        wait(500)
        input = webDriver.findElementByName("input")
        submit = webDriver.findElementByName("submitButton")
        input.sendKeys("\b\b\bA-B\nB?A")
        wait(200)

        submit.click()
```

```
            wait(200)
            output = webDriver.findElementByName("output")
            output.text.trim() shouldBe "[A]\n[B, A]"
            wait(500)
            input = webDriver.findElementByName("input")
            submit = webDriver.findElementByName("submitButton")
            input.sendKeys("\b\b\bA?B")

            wait(200)
            submit.click()
            wait(200)
            output = webDriver.findElementByName("output")
            output.text.trim() shouldBe "[A]\n[A, B]"
            wait(500)
            input = webDriver.findElementByName("input")
            submit = webDriver.findElementByName("submitButton")
            input.sendKeys("\nA?C")

            wait(200)
            submit.click()
            wait(200)
            output = webDriver.findElementByName("output")
            output.text shouldMatch Regex("VertexNot.*")
            wait(500)
            input = webDriver.findElementByName("input")
            submit = webDriver.findElementByName("submitButton")

            input.sendKeys("\b\b\bA(1,1)")
            wait(200)
            submit.click()
            wait(200)
            output = webDriver.findElementByName("output")
            output.text shouldMatch Regex("VertexAlready.*")
            wait(500)
            input = webDriver.findElementByName("input")

            submit = webDriver.findElementByName("submitButton")
            input.sendKeys("****")
            wait(200)
            submit.click()
            wait(200)
            output = webDriver.findElementByName("output")
            output.text shouldMatch Regex("GraphParse.*")
            webDriver.close()
            System.exit(0)
        }
    }
```

Листинг 1 — Main.kt

Листинг 2 — Window.kt

```
import java.awt.*
import java.awt.event.ActionListener
import java.io.PrintWriter
import java.io.StringWriter
import java.util.*
import javax.swing.*
import kotlin.concurrent.thread

class Step(val input: String) {
    companion object {
        private var superNumber = 0
    }

    val serial = ++superNumber
    val error: Boolean
    val output: String

    init {
        var error = false
        var output: String
        try {
            output = graphFromString(input)
        } catch (e: Exception) {
            val sw = StringWriter()
            e.printStackTrace(PrintWriter(sw))
            output = sw.toString()
            error = true
        }
        this.output = output
        this.error = error
    }
}


object Window : JFrame("Path finding") {

    private fun <T : Component>T.name(str: String): T {
        this.name = str
        return this
    }

    private var step = Step("A(1, 1)\n" +
            "B(1, -1)\n" +
            "C(-1, 1)\n" +
            "D(-1, -1)\n" +
            "\n" +
            "A - B\n" +
            "A - C\n" +
            "B - C\n" +
            "C - D\n" +
            "\n" +
            "A ? B\n" +
            "A ? C\n" +
            "A ? D")
    private val inputTextArea = JTextArea(step.input).name("inputTextArea")
    private val outputTextArea = JTextArea(step.output).name("outputTextArea")
    private val previousSteps = Stack<Step>()
    private val leftScrollPane = JScrollPane(inputTextArea).name("leftScrollPane")
    private val rightScrollPane = JScrollPane(outputTextArea).name("rightScrollPane")
    private val convertButton = JButton("Submit").name("convertButton")
    private val undoButton = JButton("Undo").name("undoButton")
    private val buttons = listOf(convertButton, undoButton)
    private val jPanel = JPanel()
    private val bPanel = JPanel()
        private val l = GridLayout(1, 3)

    init {
```

```kotlin
        size = Dimension(750, 680)

        contentPane = jPanel

        layout = l

        add(leftScrollPane)

        bPanel.layout = FlowLayout()

        bPanel.add(convertButton)
        bPanel.add(undoButton)

        add(bPanel)

        add(rightScrollPane)

        fun setStep(step: Step) {
            outputTextArea.foreground =
                    if (step.error)
                        Color.RED
                    else
                        this.foreground
            outputTextArea.text = step.output
        }

        convertButton.addActionListener {
            buttons.forEach { it.isEnabled = false }
            thread {
                val step = Step(inputTextArea.text)
                setStep(step)
                synchronized(previousSteps) {
                    previousSteps.push(Window.step)
                }
                Window.step = step
                buttons.forEach { it.isEnabled = true }
            }
        }

        undoButton.addActionListener {
            buttons.forEach { it.isEnabled = false }
            var step: Step? = null
            synchronized(previousSteps) {
                if (previousSteps.isNotEmpty())
                    step = previousSteps.pop()
            }
            val immutableStep = step
            if (immutableStep != null) {
                inputTextArea.text = immutableStep.input
                setStep(immutableStep)
                Window.step = immutableStep
            }
            buttons.forEach { it.isEnabled = true }
        }

        outputTextArea.isEditable = false

        isVisible = true
```

```
    }
}

fun main(args: Array<String>) {
    Window
}
```

Листинг 2 — Window.kt

Листинг 3 — WebTest.kt

```
import com.sun.net.httpserver.HttpExchange
import com.sun.net.httpserver.HttpHandler
import com.sun.net.httpserver.HttpServer
import org.openqa.selenium.chrome.ChromeDriver
import java.io.File
import java.io.PrintWriter
import java.io.StringWriter
import java.lang.Exception
import java.net.InetSocketAddress
import java.net.URLDecoder

object WebTest : HttpHandler {
    init {
        val server = HttpServer.create(InetSocketAddress(8000), 0)
        server.createContext("/test", WebTest)
        server.executor = null
        server.start()
    }

    override fun handle(t: HttpExchange) {
        var inputText = "A(1, 1)\n" +
                "B(1, -1)\n" +
                "C(-1, 1)\n" +
                "D(-1, -1)\n" +
                "\n" +
                "A - B\n" +
                "A - C\n" +
                "B - C\n" +
                "C - D\n" +
                "\n" +
                "A ? B\n" +
                "A ? C\n" +
                "A ? D"
        var outputText = "[A, B]\n" +
                "[A, C]\n" +
                "[A, C, D]"
        var color = "black"

        try {
            val request = URLDecoder.decode(t.requestURI
                    .toString()
                    .split("test?input=")[1],
                    "UTF8")
            inputText = request
            val result = graphFromString(request)
            outputText = result
        } catch (e: IndexOutOfBoundsException) {

        } catch (e: Exception) {
            e.printStackTrace()
```

```
                val sw = StringWriter()
                e.printStackTrace(PrintWriter(sw))
                color = "red"
                outputText = sw.toString()
            }

            val response = "<!DOCTYPE html>\n" +
                    "<html>\n" +
                    "<head>\n" +
                    "    <meta charset=\"UTF-8\">\n" +
                    "    <title>Test</title>\n" +
                    "    <style>\n" +
                    "        textarea, form, input {\n" +
                    "            display: inline;\n" +
                    "        }\n" +
                    "\n" +
                    "        textarea {\n" +
                    "            width: calc(50% - 100px);\n" +
                    "            height: 100%;\n" +
                    "            resize: none;\n" +
                    "        }\n" +
                    "\n" +
                    "        .black {\n" +
                    "            color: black;\n" +
                    "        }\n" +
                    "\n" +
                    "        .red {\n" +
                    "            color: red;\n" +
                    "        }\n" +
                    "    </style>\n" +
                    "    <script type=\"text/javascript\">\n" +
                    "        var doc = [];\n" +
                    "        \n" +
                    "        function getDoc() {\n" +
                    "            //alert(\"READING COOKIE: \" + document.cookie);\n" +
                    "            try {\n" +
                    "                var d = JSON.parse(document.cookie);\n" +
                    "                if (Array.isArray(d))\n" +
                    "                    doc = doc;\n" +
                    "                else\n" +
                    "                    doc = [];\n" +
                    "            } catch (e) {\n" +
                    "                doc = [];\n" +
                    "                document.cookie = \"[]\";\n" +
                    "            }\n" +
                    "        }\n" +
                    "\n" +
                    "        function putDoc() {\n" +
                    "            document.cookie = JSON.stringify(doc);\n" +
                    "\n" +
                    "            //alert(\"WRITING COOKIE: \" + document.cookie);\n" +
                    "        }\n" +
                    "\n" +
                    "        function submit() {\n" +
                    "            getDoc();\n" +
                    "            doc.push({\n" +

                    ↪   "                input: document.getElementsByName(\"input\")[0].value,\n"
                    ↪   +

                    ↪   "                output: document.getElementsByName(\"output\")[0].value,\n"
                    ↪   +

                    ↪   "                error: document.getElementsByName(\"output\")[0].className\
                    ↪   +
                    "            });\n" +
                    "            putDoc();\n" +
                    "            var xhr = new XMLHttpRequest();\n" +

                    ↪   "            var url = 'test?input=' + encodeURI(document.getElementsByName(
                    ↪   +
                    "            xhr.open('GET', url);\n" +
                    "            xhr.onload = function() {\n" +
                    "                if (xhr.status === 200) {\n" +

                    ↪   "                    window.history.pushState({\"html\":url,\"pageTitle\":\"
                    ↪   +
                    "                    document.open();\n" +
                    "                    document.write(xhr.responseText);\n" +
                    "                    document.close();\n" +
                    "                }\n" +
                    "                else {\n" +

                    ↪   "                    alert('Request failed.  Returned status of ' + xhr.stat
                    ↪   +
                    "                }\n" +
                    "            };\n" +
                    "            xhr.send();\n" +
                    "        }\n" +
                    "\n" +
                    "        function undo() {\n" +
                    "            alert(\"Undo\");\n" +
                    "\n" +
                    "            getDoc();\n" +
                    "\n" +
                    "            if (doc.length > 0) {\n" +
                    "                var last = doc.pop();\n" +
                    "                putDoc();\n" +
                    "                var url = 'test?input=' + encodeURI(last.input);\n" +

                    ↪   "                window.history.pushState({\"html\":url,\"pageTitle\":\"Test

                    ↪   +

                    ↪   "                document.getElementsByName(\"input\")[0].value = last.input
                    ↪   +

                    ↪   "                document.getElementsByName(\"output\")[0].value = last.outp
                    ↪   +

                    ↪   "                document.getElementsByName(\"output\")[0].className = last.
                    ↪   +
                    "            }\n" +
                    "            document.cookie = JSON.stringify(doc);\n" +
                    "        }\n" +
                    "    </script>\n" +
                    "</head>\n" +
                    "<body>\n" +
                    "    <textarea name=\"input\">$inputText</textarea>\n" +
                    "    <div>\n" +

                    ↪  "        <button name=\"submitButton\" type=\"submit\" onclick=\"submit()\">
                    ↪  +

                    ↪  "        <!-- <button type=\"submit\" onclick=\"undo()\">undo</button> -->\n
                    ↪  +
                    "    </div>\n" +

                    ↪  "    <textarea name=\"output\" class=\"$color\" readonly>$outputText</textar
                    ↪  +
                    "</body>\n" +
                    "</html>"
        t.sendResponseHeaders(200, response.length.toLong())
        val os = t.getResponseBody()
        os.write(response.toByteArray())
        os.close()
    }
}

fun main(args: Array<String>) {
    WebTest

}
```

Листинг 3 — WebTest.kt

Листинг 4 — AStar.kt

```
fun aStar(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {

    val frontier = PriorityQueue(from)
    val cameFrom = Array<Graph.Vertex?>(from.graph.size) { null }
    val handled = BooleanArray(from.graph.size)
    val costSoFar = DoubleArray(from.graph.size) { 0.0 }

    handled[from.i] = true

    fun directDist(v: Graph.Vertex) =
            Math.sqrt(
                    Math.pow((to.y - v.y), 2.0) +
                            Math.pow((to.x - v.x), 2.0)
            )

    var last = from
    while (frontier.isNotEmpty()) {
        val current = frontier.pop()
        last = current

        if (current == to)
            break

        current.linkedVertices.forEach {
            val newCost = costSoFar[current.i] + it.distanceTo(current)
            if (!handled[it.i] || newCost < costSoFar[it.i]) {
                handled[it.i] = true
                costSoFar[it.i] = newCost
```

```kotlin
                cameFrom[it.i] = current
                val priority = newCost + directDist(it)
                frontier.push(it, priority)
            }
        }
    }

    if (last == to) {
        val ml = mutableListOf<Graph.Vertex>()
        var curr: Graph.Vertex? = last
        while (curr != null) {
            ml.add(curr)
            curr = cameFrom[curr.i]
        }
        //return ml.toList()
        return ml.reversed()
    }

    return listOf()
}
```

Листинг 4 — AStar.kt

Листинг 5 — Graph.kt

```kotlin
class Graph(vertexBuilders: List<GraphBuilder.VertexBuilder>) {
    companion object {
        class GraphBuilder(initializer: (GraphBuilder) -> Unit = {}) {

            class VertexBuilder(val name: String, private val i: Int, val x: Double, val y: Double) {
                private val links = mutableSetOf<VertexBuilder>()

                fun link(other: VertexBuilder) {
                    synchronized(this) {
                        links.add(other)
                    }
                    synchronized(other) {
                        other.links.add(this)
                    }
                }

                override fun hashCode(): Int = i

                override fun equals(other: Any?): Boolean =
                        this === other

                operator fun minus(other: VertexBuilder) {
                    link(other)
                }

                fun getLinks(size: Int): DoubleArray {
                    val ml = DoubleArray(size) { -1.0 }
                    links.forEach {
                        val i = it.i
                        if (i in 0..(size - 1))
                            ml[i] = Math.sqrt(
                                    Math.pow((x - it.x).toDouble(), 2.0) +
                                            Math.pow((y - it.y).toDouble(), 2.0)
                            )
                    }
                    return ml
                }
            }

            private val vertices = mutableListOf<VertexBuilder>()

            init {
                initializer(this)
            }

            fun vertex(name: String, x: Double, y: Double): VertexBuilder {
                synchronized(this) {
                    val vertex = VertexBuilder(name, vertices.size, x, y)
                    vertices.add(vertex)
                    return vertex
                }
            }

            fun vertex(name: String, x: Int, y: Int) = vertex(name, x.toDouble(), y.toDouble())

            fun build(): Graph {
                synchronized(this) {
                    return Graph(vertices.toList())
                }
            }
        }

        fun build(initializer: (GraphBuilder) -> Unit) = GraphBuilder(initializer).build()
    }

    data class Vertex(val name: String, val graph: Graph, val i: Int, val x: Double, val y: Double, private val linkArray: DoubleArray) {
        override fun hashCode(): Int = i
        override fun equals(other: Any?): Boolean {
            if (this === other)
                return true
            if (other !is Vertex)
                return false
            if (
                    graph == other.graph &&
                    i == other.i &&
                    x == other.x &&
                    y == other.y &&
                    linkArray.contentEquals(other.linkArray)
            )
                return true
            return false
        }

        val linkedVertices by lazy {
            val ml = mutableListOf<Vertex>()
            graph.forEach {
                if (this links it)
                    ml.add(it)
            }
             ml.toList()
        }

        infix fun links(other: Vertex): Boolean {
            if (other.graph === graph && other.i >= 0 && other.i < linkArray.size)
                return linkArray[other.i] != -1.0
            return false
        }

        infix fun distanceTo(other: Vertex): Double {
            if (other.graph === graph && other.i >= 0 && other.i < linkArray.size)
                return linkArray[other.i]
            throw IllegalStateException()
        }

        override fun toString(): String = name
    }

    private val vertices: List<Vertex>
    val size: Int

    init {
        vertices = List(vertexBuilders.size) { i ->
            val it = vertexBuilders[i]
            Vertex(
                    it.name,
                    this,
                    i,
                    it.x,
                    it.y,
                    it.getLinks(vertexBuilders.size)
            )
        }
        size = vertices.size
    }

    operator fun get(i: Int): Vertex {
        if (i in 0..(size - 1))
            return  vertices[i]
        throw ArrayIndexOutOfBoundsException(i)
    }

    operator fun get(name: String): Vertex {
        var i = 0
        do {
            if (vertices[i].name == name)
                return vertices[i]
        } while (++i < size)
        throw NullPointerException(name)
    }

    fun forEach(action: (Vertex) -> Unit) {
        var i = 0
        do {
            action(vertices[i])
        } while (++i < size)
    }
}
```

Листинг 5 — Graph.kt

Листинг 6 — LoadGraph.kt

```kotlin
import org.w3c.dom.Document
import org.w3c.dom.Element
import java.util.zip.ZipFile
import javax.xml.parsers.DocumentBuilderFactory


fun loadGraph(path: String): Graph {
    val zipFile = ZipFile(path)
    val entries = zipFile.entries()

    fun parseGraph(doc: Document): Graph {
        return Graph.build {
            val vertices = hashMapOf<String, Graph.Companion.GraphBuilder.
            ↪  VertexBuilder>()

            var res = doc.getElementsByTagName("element")
            points@ for (i in 0..(res.length - 1)) {
                val node = res.item(i) as Element
                if (node.getAttribute("type") == "point") {
                    val name = node.getAttribute("label")
                    val children = node.getElementsByTagName("coords")
                    for (j in 0..(children.length - 1)) {
                        val subNode = children.item(j) as Element
                        val x = subNode.getAttribute("x").toDouble()
                        val y = subNode.getAttribute("y").toDouble()
                        vertices[name] = it.vertex(name, x, y)
                        continue@points
                    }
                }
            }
            res = doc.getElementsByTagName("command")
            fragments@ for (i in 0..(res.length - 1)) {
                val node = res.item(i) as Element
                if (node.getAttribute("name") == "Segment") {
                    val children = node.getElementsByTagName("input")
                    for (j in 0..(children.length - 1)) {
                        val subNode = children.item(j) as Element
                        val a0 = vertices[subNode.getAttribute("a0")]
                        val a1 = vertices[subNode.getAttribute("a1")]
                        if (a0 != null && a1 != null)
                            a0 - a1
                        continue@fragments
                    }
                }
            }
        }
    }

    while (entries.hasMoreElements()) {
        val entry = entries.nextElement()
        if (entry.name == "geogebra.xml") {
            val stream = zipFile.getInputStream(entry)
            val dbFactory = DocumentBuilderFactory.newInstance()
            val dBuilder = dbFactory.newDocumentBuilder()
            val doc = dBuilder.parse(stream)
            return parseGraph(doc)
        }
    }

    return Graph.build { }
}
```

Листинг 6 — LoadGraph.kt

Листинг 7 — PriorityQueue.kt

```
import java.util.*

class PriorityQueue<T>(vararg ts: T) {

    private val mp = mutableListOf<Double>()
    private val mt = mutableListOf<T>()

    init {
        ts.forEach {
            push(it, 0.0)
        }
    }
    fun push(t: T, priority: Double) {
        synchronized(this) {
            mp.add(priority)
            mt.add(t)
        }
    }

    fun isEmpty() = mp.isEmpty()

    fun isNotEmpty() = !isEmpty()

    fun pop(): T {
        if (isEmpty())
            throw EmptyStackException()
        synchronized(this) {
            var max = mp[0]
            var index = 0
            var i = 0
            do {
                if (mp[i] < max)
                    index = i
            } while (++i < mp.size)
            mp.removeAt(index)
            val result = mt[index]
            mt.removeAt(index)
            return result
        }
    }
}
```

Листинг 7 — PriorityQueue.kt