

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

```
1  import org.assertj.swing.fixture.FrameFixture
2  import org.openqa.selenium.chrome.ChromeDriver
3  import java.awt.Dimension
4  import java.io.File
5  import java.util.*
6
7  val debug = true
8
9  fun main(args: Array<String>) {
10
11      fun maybePrint(any: Any) {
12          if (debug)
13              println(any)
14      }
15
16      fun maybeError(any: Any) {
17          if (debug)
18              System.err.println(any)
19      }
20
21      for (i in 0..(if (debug) 1 else 10000))
22          Tests.javaClass.declaredMethods.forEach {
23              if (it.isAnnotationPresent(Test::class.java)) {
24                  try {
25                      val tests = it.invoke(Tests)
26                      if (it.returnType.canonicalName == "void" ||
27                          (it.returnType.canonicalName == "boolean" &&
28                              tests as Boolean))
29                          maybePrint("${it.name} passed")
30                      else
31                          maybeError("${it.name} error")
32                  } catch (e: Exception) {
33                      maybeError("${it.name} error")
34                      if (debug)
35                          e.printStackTrace()
36                  }
37              }
38          }
```

```

38         }
39     }
40
41     infix fun String.shouldMatch(regex: Regex) {}
42
43     infix fun <T, V> T.shouldBe(other: V) {
44         if (this != other)
45             throw TestException("$this doesn't equal $other")
46     }
47
48     class TestException(msg: String) : RuntimeException(msg)
49
50     @Target(AnnotationTarget.FUNCTION)
51     annotation class Test
52
53     object Tests {
54         fun loadWayAsString(name: String, from: String, to: String): String {
55             val graph = loadGraph("$name.ggb")
56             val way = aStar(graph[from], graph[to])
57             return way.toString()
58         }
59
60         @Test
61         fun testLongWay2(): Boolean {
62             return loadWayAsString("Graph4", "A", "Z") ==
63                 ↪ "[A, B, S, T, R, E, M, N, Q, P, Z]"
64         }
65
66         @Test
67             /** Test:
68              * if (current == to) -> true
69              * if (last == to) -> true
70              */
71         fun testB1() {
72             val graph = Graph.build {
73                 val a = it.vertex("a", 1, 1)
74             }
75             aStar(graph["a"], graph["a"]) shouldBe listOf(graph["a"])
76         }
77
78         @Test
79             /** Test:
80              * if (current == to) -> false
81              * if (!handled[it.i] || newCost < costSoFar[it.i]) -> true
82              * if (current == to) -> true
83              * if (last == to) -> true
84              */
85         fun testB2() {
86             val graph = Graph.build {
87                 val a = it.vertex("a", 1, 1)
88                 val b = it.vertex("b", -1, -1)
89
90                 a - b
91             }
92             aStar(graph["a"], graph["b"]) shouldBe listOf(graph["a"], graph["b"])
93         }
94
95         @Test

```

```

95     /** Test:
96      * if (current == to) -> false
97      * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
98      * if (!handled[it.i] || newCost < costSoFar[it.i]) -> true
99      * if (current == to) -> true
100     * if (last == to) -> true
101     */
102    fun testB3() {
103        val graph = Graph.build {
104            val a = it.vertex("a", 1, 1)
105            val b = it.vertex("b", -1, -1)
106            val c = it.vertex("c", -2, -2)
107
108            a - b
109            a - c
110        }
111        aStar(graph["a"], graph["c"]) shouldBe listOf(graph["a"], graph["c"])
112    }
113
114    @Test
115    /** Test:
116     * if (current == to) -> false
117     * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
118     * if (!handled[it.i] || newCost < costSoFar[it.i]) -> false
119     * while (frontier.isNotEmpty()) -> false
120     * if (last == to) -> false
121     */
122    fun testB4() {
123        val graph = Graph.build {
124            val a = it.vertex("a", 1, 1)
125            val b = it.vertex("b", -1, -1)
126            val c = it.vertex("c", -2, -2)
127
128            a - b
129        }
130        aStar(graph["a"], graph["c"]) shouldBe listOf<Graph.Vertex>()
131    }
132
133    @Test
134    fun testA1() {
135        val graph = Graph.build {
136            val a = it.vertex("a", 0, 1)
137            val b = it.vertex("b", 1, 0)
138            val c = it.vertex("c", 0, 0)
139            val d = it.vertex("d", 0, -1)
140
141            a - b
142            a - c
143
144            d - b
145            d - c
146        }
147        aStar(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>(graph["a"], graph[
148            ↪ "c"], graph["d"])
149    }
150
151    @Test
152    fun testA2() {

```

```

152     val graph = Graph.build {
153         val a = it.vertex("a", 0, 1)
154         val b = it.vertex("b", 0, 0)
155         val c = it.vertex("c", -1, 0)
156         val d = it.vertex("d", 0, -1)
157
158         a - b
159         a - c
160
161         d - b
162         d - c
163     }
164     aStar(graph["a"], graph["d"]) shouldBe listOf<Graph.Vertex>(graph["a"], graph[
165         ↪ "b"], graph["d"])
166 }
167
168 // @Test
169 fun testC1() {
170     if (!debug)
171         return
172
173     val max = 40
174     val rn = Random()
175     val sb = StringBuilder()
176     for (i in 1..max) {
177         for (j in 1..max)
178             sb.appendln("val${i}n$j($i, $j)")
179         if (i > 1)
180             for (j in 1..max) {
181                 for (k in 1..max)
182                     if (rn.nextInt(2) == 0)
183                         sb.appendln("val${i - 1}n$j - val${i}n$k")
184             }
185         for (j in 1..max) {
186             for (k in 1..max)
187                 if (rn.nextInt(2) == 0)
188                     sb.appendln("val${i}n$j - val${i}n$k")
189         }
190     }
191     for (i in 1..max)
192         sb.appendln("val1n$i ? val${max}n${max + 1 - i}")
193     val g = graphFromString(sb.toString())
194 }
195
196 @Test
197 fun windowTest() {
198     if (!debug)
199         return
200
201     fun wait(ms: Int) {
202         Thread.sleep(ms.toLong())
203     }
204
205     val fixture = FrameFixture(Window)
206     for (i in 0..4) {
207         when (i) {
208             1 -> fixture.resizeTo(Dimension(1000, 200))
209             2 -> fixture.resizeTo(Dimension(200, 1000))

```

```

209         3 -> fixture.resizeTo(Dimension(200, 200))
210         4 -> fixture.resizeTo(Dimension(100, 100))
211     }
212
213     if (i > 0) {
214         fixture.textBox("inputTextArea").setText("A(1,1)\nA?A")
215         fixture.button("convertButton").click()
216         fixture.textBox("outputTextArea").requireText("[A]")
217         fixture.button("undoButton").click()
218         continue
219     }
220
221     fixture.textBox("inputTextArea").setText("")
222     fixture.textBox("inputTextArea").enterText("A(1,1)\nA?A")
223     fixture.button("convertButton").click()
224     fixture.textBox("outputTextArea").requireText("[A]")
225     ///
226     fixture.textBox("inputTextArea").setText("")
227     fixture.textBox("inputTextArea").setText("A(1,1)\nB(2,2)\nA?B")
228     fixture.button("convertButton").click()
229     fixture.textBox("outputTextArea").requireText("[]")
230     ///
231     fixture.textBox("inputTextArea").setText("")
232     fixture.textBox("inputTextArea").setText("A(1,1)\nB(2,2)\nA-B\nA?B")
233     fixture.button("convertButton").click()
234     fixture.textBox("outputTextArea").requireText(Regex("\\[A,. *]").toPattern())
235     ///
236     fixture.textBox("inputTextArea").setText("")
237     fixture.textBox("inputTextArea").setText("A(1,1)\nA-B")
238     fixture.button("convertButton").click()
239     wait(500)
240     fixture.textBox("outputTextArea").requireText(
241         "VertexNotDeclaredException: Vertex \"B\" isn't declared!\n" +
242
243         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
244         ↪ +
245
246         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
247         ↪ +
248
249         ↪ "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
250         ↪ +
251         ↪ "\tat Graph\$Companion.build(Graph.kt:65)\n" +
252         ↪ "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
253         ↪ +
254         ↪ "\tat Step.<init>(Window.kt:22)\n" +
255         ↪ "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
256         ↪ "\tat Window\$2\$2.invoke(Window.kt:35)\n" +
257
258         ↪ "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
259
260     )
261     ///
262     fixture.textBox("inputTextArea").setText("")
263     fixture.textBox("inputTextArea").setText("A(1,1)\nA(1,1)")
264     fixture.button("convertButton").click()
265     wait(500)
266     fixture.textBox("outputTextArea").requireText(

```

```

258         "VertexAlreadyDeclaredException: Vertex \"A\" already declared!\n"
259         ↪ +
260
261         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
262         ↪ +
263         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
264         ↪ +
265         ↪ "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
266         ↪ +
267         "\tat Graph\$Companion.build(Graph.kt:65)\n" +
268         "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
269         ↪ +
270         "\tat Step.<init>(Window.kt:22)\n" +
271         "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
272         "\tat Window\$2\$2.invoke(Window.kt:35)\n" +
273         ↪ "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
274     )
275     ///
276     fixture.textBox("inputTextArea").setText("")
277     fixture.textBox("inputTextArea").setText("A*B")
278     fixture.button("convertButton").click()
279     wait(500)
280     fixture.textBox("outputTextArea").requireText(
281         "GraphParserException: Line \"A*B\" not recognized.\n" +
282         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
283         ↪ +
284         ↪ "\tat StringGraphKt\$graphFromString\$graph\$1.invoke(StringGraph
285         ↪ +
286         ↪ "\tat Graph\$Companion\$GraphBuilder.<init>(Graph.kt:45)\n"
287         ↪ +
288         "\tat Graph\$Companion.build(Graph.kt:65)\n" +
289         "\tat StringGraphKt.graphFromString(StringGraph.kt:10)\n"
290         ↪ +
291         "\tat Step.<init>(Window.kt:22)\n" +
292         "\tat Window\$2\$2.invoke(Window.kt:97)\n" +
293         "\tat Window\$2\$2.invoke(Window.kt:35)\n" +
294         ↪ "\tat kotlin.concurrent.ThreadsKt\$thread\$thread\$1.run(Thread.
295     )
296     fixture.textBox("inputTextArea").setText("")
297     fixture.textBox("inputTextArea").setText("A(1,1)\nB(1,2)\nA-B\nA?B")
298     fixture.button("convertButton").click()
299     fixture.textBox("outputTextArea").requireText("[A, B]")
300
301     fixture.textBox("inputTextArea").setText("")
302     fixture.textBox("inputTextArea").setText("A(1,1)\nB(1,2)\nA-B\nB?A")
303     fixture.button("convertButton").click()
304     fixture.textBox("outputTextArea").requireText("[B, A]")
305
306     fixture.button("undoButton").click()
307     fixture.textBox("inputTextArea").requireText("A(1,1)\nB(1,2)\nA-B\nA?B")
308     fixture.textBox("outputTextArea").requireText("[A, B]")

```

```

299
300         for (j in 0..20) {
301             fixture.button("undoButton").click()
302             wait(50)
303         }
304     }
305
306     Window.isVisible = false
307     /*
308     fixture.textBox("outputTextArea").requireText(
309         Regex("GraphParser.+").toPattern()
310     )
311     */
312 }
313
314 @Test
315 fun webTest() {
316     if (!debug)
317         return
318
319     WebTest
320
321     fun wait(ms: Int) {
322         Thread.sleep(ms.toLong())
323     }
324
325     val webDriver = ChromeDriver()
326     System.setProperty("webdriver.chrome.driver", File("chromedriver").
327         ↪ absolutePath);
328     webDriver.navigate().to("http://localhost:8000/test?input=")
329     wait(200)
330
331     var input = webDriver.findElementByName("input")
332     var submit = webDriver.findElementByName("submitButton")
333     input.sendKeys("A(1,1)\nA?A")
334     wait(200)
335     submit.click()
336     wait(200)
337
338     var output = webDriver.findElementByName("output")
339     output.text.trim() shouldBe "[A]"
340     wait(500)
341     input = webDriver.findElementByName("input")
342     submit = webDriver.findElementByName("submitButton")
343     input.sendKeys("\nB(1,1)\nA?B")
344     wait(200)
345     submit.click()
346
347     wait(200)
348     output = webDriver.findElementByName("output")
349     output.text.trim() shouldBe "[A]\n[]"
350     wait(500)
351     input = webDriver.findElementByName("input")
352     submit = webDriver.findElementByName("submitButton")
353     input.sendKeys("\b\b\bA-B\nB?A")
354     wait(200)
355     submit.click()

```



```

356     wait(200)
357     output = webDriver.findElementByName("output")
358     output.text.trim() shouldBe "[A]\n[B, A]"
359     wait(500)
360     input = webDriver.findElementByName("input")
361     submit = webDriver.findElementByName("submitButton")
362     input.sendKeys("\b\b\bA?B")
363
364     wait(200)
365     submit.click()
366     wait(200)
367     output = webDriver.findElementByName("output")
368     output.text.trim() shouldBe "[A]\n[A, B]"
369     wait(500)
370     input = webDriver.findElementByName("input")
371     submit = webDriver.findElementByName("submitButton")
372     input.sendKeys("\nA?C")
373
374     wait(200)
375     submit.click()
376     wait(200)
377     output = webDriver.findElementByName("output")
378     output.text shouldMatch Regex("VertexNot.*")
379     wait(500)
380     input = webDriver.findElementByName("input")
381     submit = webDriver.findElementByName("submitButton")
382
383     input.sendKeys("\b\b\bA(1,1)")
384     wait(200)
385     submit.click()
386     wait(200)
387     output = webDriver.findElementByName("output")
388     output.text shouldMatch Regex("VertexAlready.*")
389     wait(500)
390     input = webDriver.findElementByName("input")
391
392     submit = webDriver.findElementByName("submitButton")
393     input.sendKeys("****")
394     wait(200)
395     submit.click()
396     wait(200)
397     output = webDriver.findElementByName("output")
398     output.text shouldMatch Regex("GraphParse.*")
399     webDriver.close()
400     System.exit(0)
401 }
402 }

```

Листинг 1 — Main.kt

Листинг 2 — Window.kt

```

1  import java.awt.*
2  import java.awt.event.ActionListener
3  import java.io.PrintWriter
4  import java.io.StringWriter
5  import java.util.*
6  import javax.swing.*
7  import kotlin.concurrent.thread
8
9  class Step(val input: String) {

```

```

10 companion object {
11     private var superNumber = 0
12 }
13
14 val serial = ++superNumber
15 val error: Boolean
16 val output: String
17
18 init {
19     var error = false
20     var output: String
21     try {
22         output = graphFromString(input)
23     } catch (e: Exception) {
24         val sw = StringWriter()
25         e.printStackTrace(PrintWriter(sw))
26         output = sw.toString()
27         error = true
28     }
29     this.output = output
30     this.error = error
31 }
32 }
33
34
35 object Window : JFrame("Path finding") {
36
37     private fun <T : Component>T.name(str: String): T {
38         this.name = str
39         return this
40     }
41
42     private var step = Step("A(1, 1)\n" +
43         "B(1, -1)\n" +
44         "C(-1, 1)\n" +
45         "D(-1, -1)\n" +
46         "\n" +
47         "A - B\n" +
48         "A - C\n" +
49         "B - C\n" +
50         "C - D\n" +
51         "\n" +
52         "A ? B\n" +
53         "A ? C\n" +
54         "A ? D")
55
56     private val inputTextArea = JTextArea(step.input).name("inputTextArea")
57     private val outputTextArea = JTextArea(step.output).name("outputTextArea")
58     private val previousSteps = Stack<Step>()
59     private val leftScrollPane = JScrollPane(inputTextArea).name("leftScrollPane")
60     private val rightScrollPane = JScrollPane(outputTextArea).name("rightScrollPane")
61     private val convertButton = JButton("Submit").name("convertButton")
62     private val undoButton = JButton("Undo").name("undoButton")
63     private val buttons = listOf(convertButton, undoButton)
64     private val jPanel = JPanel()
65     private val bPanel = JPanel()
66         private val l = GridLayout(1, 3)
67
68     init {

```

```

68     size = Dimension(750, 680)
69
70     contentPane = jPanel1
71
72     layout = l
73
74     add(leftScrollPane)
75
76     bPanel.layout = FlowLayout()
77
78     bPanel.add(convertButton)
79     bPanel.add(undoButton)
80
81     add(bPanel)
82
83     add(rightScrollPane)
84
85     fun setStep(step: Step) {
86         outputTextArea.foreground =
87             if (step.error)
88                 Color.RED
89             else
90                 this.foreground
91         outputTextArea.text = step.output
92     }
93
94     convertButton.addActionListener {
95         buttons.forEach { it.isEnabled = false }
96         thread {
97             val step = Step(inputTextArea.text)
98             setStep(step)
99             synchronized(previousSteps) {
100                 previousSteps.push(Window.step)
101             }
102             Window.step = step
103             buttons.forEach { it.isEnabled = true }
104         }
105     }
106
107     undoButton.addActionListener {
108         buttons.forEach { it.isEnabled = false }
109         var step: Step? = null
110         synchronized(previousSteps) {
111             if (previousSteps.isNotEmpty())
112                 step = previousSteps.pop()
113         }
114         val immutableStep = step
115         if (immutableStep != null) {
116             inputTextArea.text = immutableStep.input
117             setStep(immutableStep)
118             Window.step = immutableStep
119         }
120         buttons.forEach { it.isEnabled = true }
121     }
122
123     outputTextArea.isEditable = false
124
125     isVisible = true

```

```

126     }
127 }
128
129 fun main(args: Array<String>) {
130     Window
131 }

```

Листинг 2 — Window.kt

Листинг 3 — WebTest.kt

```

1  import com.sun.net.httpserver.HttpExchange
2  import com.sun.net.httpserver.HttpHandler
3  import com.sun.net.httpserver.HttpServer
4  import org.openqa.selenium.chrome.ChromeDriver
5  import java.io.File
6  import java.io.PrintWriter
7  import java.io.StringWriter
8  import java.lang.Exception
9  import java.net.InetSocketAddress
10 import java.net.URLDecoder
11
12 object WebTest : HttpHandler {
13     init {
14         val server = HttpServer.create(InetSocketAddress(8000), 0)
15         server.createContext("/test", WebTest)
16         server.executor = null
17         server.start()
18     }
19
20     override fun handle(t: HttpExchange) {
21         var inputText = "A(1, 1)\n" +
22             "B(1, -1)\n" +
23             "C(-1, 1)\n" +
24             "D(-1, -1)\n" +
25             "\n" +
26             "A - B\n" +
27             "A - C\n" +
28             "B - C\n" +
29             "C - D\n" +
30             "\n" +
31             "A ? B\n" +
32             "A ? C\n" +
33             "A ? D"
34         var outputText = "[A, B]\n" +
35             "[A, C]\n" +
36             "[A, C, D]"
37         var color = "black"
38
39         try {
40             val request = URLDecoder.decode(t.requestURI
41                 .toString()
42                 .split("test?input=")[1],
43                 "UTF8")
44             inputText = request
45             val result = graphFromString(request)
46             outputText = result
47         } catch (e: IndexOutOfBoundsException) {
48
49         } catch (e: Exception) {
50             e.printStackTrace()

```

```

51     val sw = StringWriter()
52     e.printStackTrace(PrintWriter(sw))
53     color = "red"
54     outputText = sw.toString()
55 }
56
57 val response = "<!DOCTYPE html>\n" +
58     "<html>\n" +
59     "<head>\n" +
60     "    <meta charset=\"UTF-8\">\n" +
61     "    <title>Test</title>\n" +
62     "    <style>\n" +
63     "        textarea, form, input {\n" +
64     "            display: inline;\n" +
65     "        }\n" +
66     "\n" +
67     "        textarea {\n" +
68     "            width: calc(50% - 100px);\n" +
69     "            height: 100%;\n" +
70     "            resize: none;\n" +
71     "        }\n" +
72     "\n" +
73     "        .black {\n" +
74     "            color: black;\n" +
75     "        }\n" +
76     "\n" +
77     "        .red {\n" +
78     "            color: red;\n" +
79     "        }\n" +
80     "</style>\n" +
81     "<script type=\"text/javascript\">\n" +
82     "    var doc = [];\n" +
83     "    \n" +
84     "    function getDoc() {\n" +
85     "        //alert(\"READING COOKIE: \" + document.cookie);\n" +
86     "        try {\n" +
87     "            var d = JSON.parse(document.cookie);\n" +
88     "            if (Array.isArray(d))\n" +
89     "                doc = doc;\n" +
90     "            else\n" +
91     "                doc = [];\n" +
92     "        } catch (e) {\n" +
93     "            doc = [];\n" +
94     "            document.cookie = \"[]\";\n" +
95     "        }\n" +
96     "    }\n" +
97     "\n" +
98     "    function putDoc() {\n" +
99     "        document.cookie = JSON.stringify(doc);\n" +
100     "\n" +
101     "        //alert(\"WRITING COOKIE: \" + document.cookie);\n" +
102     "    }\n" +
103     "\n" +
104     "    function submit() {\n" +
105     "        getDoc();\n" +
106     "        doc.push({\n"

```

```

107 ↪ "                input: document.getElementsByName(\"input\")[0].value,\n"
108 ↪ +
109 ↪ "                output: document.getElementsByName(\"output\")[0].value,\n"
110 ↪ +
111 ↪ "                error: document.getElementsByName(\"output\")[0].className\n"
112 ↪ +
113 ↪ "            });\n" +
114 ↪ "            putDoc();\n" +
115 ↪ "            var xhr = new XMLHttpRequest();\n" +
116 ↪ "            var url = 'test?input=' + encodeURIComponent(document.getElementsByName(
117 ↪ +
118 ↪ "            xhr.open('GET', url);\n" +
119 ↪ "            xhr.onload = function() {\n" +
120 ↪ "                if (xhr.status === 200) {\n" +
121 ↪ "                    window.history.pushState({\"html\":url,\"pageTitle\":\"\n"
122 ↪ +
123 ↪ "                    document.open();\n" +
124 ↪ "                    document.write(xhr.responseText);\n" +
125 ↪ "                    document.close();\n" +
126 ↪ "                }\n" +
127 ↪ "            } else {\n" +
128 ↪ "                alert('Request failed. Returned status of ' + xhr.stat
129 ↪ +
130 ↪ "                }\n" +
131 ↪ "            };\n" +
132 ↪ "            xhr.send();\n" +
133 ↪ "        }\n" +
134 ↪ "        function undo() {\n" +
135 ↪ "            alert(\"Undo\");\n" +
136 ↪ "            getDoc();\n" +
137 ↪ "            if (doc.length > 0) {\n" +
138 ↪ "                var last = doc.pop();\n" +
139 ↪ "                putDoc();\n" +
140 ↪ "                var url = 'test?input=' + encodeURIComponent(last.input);
141 ↪ +
142 ↪ "                window.history.pushState({\"html\":url,\"pageTitle\":\"Test
143 ↪ +
144 ↪ "                document.getElementsByName(\"input\")[0].value = last.input
145 ↪ +
146 ↪ "                document.getElementsByName(\"output\")[0].value = last.outp
147 ↪ +
148 ↪ "                document.getElementsByName(\"output\")[0].className = last.
149 ↪ +
150 ↪ "            }\n" +
151 ↪ "            document.cookie = JSON.stringify(doc);\n" +
152 ↪ "        }\n" +

```

```

145         "</script>\n" +
146         "</head>\n" +
147         "<body>\n" +
148         "    <textarea name=\"input\">$inputText</textarea>\n" +
149         "    <div>\n" +
150
151         ↪         "        <button name=\"submitButton\" type=\"submit\" onclick=\"submit()\">
152         ↪         +
153
154         ↪         "        <!-- <button type=\"submit\" onclick=\"undo()\">undo</button> -->\n"
155         ↪         +
156         "    </div>\n" +
157
158         ↪         "    <textarea name=\"output\" class=\"$color\" readonly>$outputText</textare
159         ↪         +
160         "</body>\n" +
161         "</html>"
162
163         t.sendResponseHeaders(200, response.length.toLong())
164         val os = t.getResponseBody()
165         os.write(response.toByteArray())
166         os.close()
167     }
168 }
169
170 fun main(args: Array<String>) {
171     WebTest
172 }

```

Листинг 3 — WebTest.kt

Листинг 4 — AStar.kt

```

1 fun aStar(from: Graph.Vertex, to: Graph.Vertex): List<Graph.Vertex> {
2
3     val frontier = PriorityQueue<Graph.Vertex>(from)
4     val cameFrom = Array<Graph.Vertex?>(from.graph.size) { null }
5     val handled = BooleanArray(from.graph.size)
6     val costSoFar = DoubleArray(from.graph.size) { 0.0 }
7
8     handled[from.i] = true
9
10    fun directDist(v: Graph.Vertex) =
11        Math.sqrt(
12            Math.pow((to.y - v.y), 2.0) +
13            Math.pow((to.x - v.x), 2.0)
14        )
15
16    var last = from
17    while (frontier.isNotEmpty()) {
18        val current = frontier.pop()
19        last = current
20
21        if (current == to)
22            break
23
24        current.linkedVertices.forEach {
25            val newCost = costSoFar[current.i] + it.distanceTo(current)
26            if (!handled[it.i] || newCost < costSoFar[it.i]) {
27                handled[it.i] = true
28                costSoFar[it.i] = newCost

```

```

29         cameFrom[it.i] = current
30         val priority = newCost + directDist(it)
31         frontier.push(it, priority)
32     }
33 }
34 }
35
36 if (last == to) {
37     val ml = mutableListOf<Graph.Vertex>()
38     var curr: Graph.Vertex? = last
39     while (curr != null) {
40         ml.add(curr)
41         curr = cameFrom[curr.i]
42     }
43     //return ml.toList()
44     return ml.reversed()
45 }
46
47 return listOf()
48 }

```

Листинг 4 — AStar.kt
Листинг 5 — Graph.kt

```

1 class Graph(vertexBuilders: List<GraphBuilder.VertexBuilder>) {
2     companion object {
3         class GraphBuilder(initializer: (GraphBuilder) -> Unit = {}) {
4
5             class VertexBuilder(val name: String, private val i: Int, val x: Double,
6                 ↪ val y: Double) {
7                 private val links = mutableSetOf<VertexBuilder>()
8
9                 fun link(other: VertexBuilder) {
10                     synchronized(this) {
11                         links.add(other)
12                     }
13                     synchronized(other) {
14                         other.links.add(this)
15                     }
16                 }
17
18                 override fun hashCode(): Int = i
19
20                 override fun equals(other: Any?): Boolean =
21                     this === other
22
23                 operator fun minus(other: VertexBuilder) {
24                     link(other)
25                 }
26
27                 fun getLinks(size: Int): DoubleArray {
28                     val ml = DoubleArray(size) { -1.0 }
29                     links.forEach {
30                         val i = it.i
31                         if (i in 0..(size - 1))
32                             ml[i] = Math.sqrt(
33                                 Math.pow((x - it.x).toDouble(), 2.0) +
34                                 Math.pow((y - it.y).toDouble(), 2.0)
35                             )
36                     }
37                 }
38             }
39         }
40     }
41 }

```



```

36         return ml
37     }
38 }
39
40 private val vertices = mutableListOf<VertexBuilder>()
41
42 init {
43     initializer(this)
44 }
45
46 fun vertex(name: String, x: Double, y: Double): VertexBuilder {
47     synchronized(this) {
48         val vertex = VertexBuilder(name, vertices.size, x, y)
49         vertices.add(vertex)
50         return vertex
51     }
52 }
53
54 fun vertex(name: String, x: Int, y: Int) = vertex(name, x.toDouble(), y.
↳ toDouble())
55
56 fun build(): Graph {
57     synchronized(this) {
58         return Graph(vertices.toList())
59     }
60 }
61
62 }
63
64 fun build(initializer: (GraphBuilder) -> Unit) = GraphBuilder(initializer).
↳ build()
65
66 }
67
68 data class Vertex(val name: String, val graph: Graph, val i: Int, val x: Double,
↳ val y: Double, private val linkArray: DoubleArray) {
69     override fun hashCode(): Int = i
70     override fun equals(other: Any?): Boolean {
71         if (this === other)
72             return true
73         if (other !is Vertex)
74             return false
75         if (
76             graph == other.graph &&
77             i == other.i &&
78             x == other.x &&
79             y == other.y &&
80             linkArray.contentEquals(other.linkArray)
81         )
82             return true
83         return false
84     }
85 }
86
87 val linkedVertices by lazy {
88     val ml = mutableListOf<Vertex>()
89     graph.forEach {
90         if (this links it)
91             ml.add(it)
92     }
93     ml.toList()

```

```

91     }
92
93     infix fun links(other: Vertex): Boolean {
94         if (other.graph === graph && other.i >= 0 && other.i < linkArray.size)
95             return linkArray[other.i] != -1.0
96         return false
97     }
98
99     infix fun distanceTo(other: Vertex): Double {
100         if (other.graph === graph && other.i >= 0 && other.i < linkArray.size)
101             return linkArray[other.i]
102         throw IllegalStateException()
103     }
104
105     override fun toString(): String = name
106 }
107
108 private val vertices: List<Vertex>
109 val size: Int
110
111 init {
112     vertices = List(vertexBuilders.size) { i ->
113         val it = vertexBuilders[i]
114         Vertex(
115             it.name,
116             this,
117             i,
118             it.x,
119             it.y,
120             it.getLinks(vertexBuilders.size)
121         )
122     }
123     size = vertices.size
124 }
125
126 operator fun get(i: Int): Vertex {
127     if (i in 0..(size - 1))
128         return vertices[i]
129     throw ArrayIndexOutOfBoundsException(i)
130 }
131
132 operator fun get(name: String): Vertex {
133     var i = 0
134     do {
135         if (vertices[i].name == name)
136             return vertices[i]
137     } while (++i < size)
138     throw NullPointerException(name)
139 }
140
141 fun forEach(action: (Vertex) -> Unit) {
142     var i = 0
143     do {
144         action(vertices[i])
145     } while (++i < size)
146 }
147 }

```

Листинг 5 — Graph.kt

```

1  import org.w3c.dom.Document
2  import org.w3c.dom.Element
3  import java.util.zip.ZipFile
4  import javax.xml.parsers.DocumentBuilderFactory
5
6
7  fun loadGraph(path: String): Graph {
8      val zipFile = ZipFile(path)
9      val entries = zipFile.entries()
10
11     fun parseGraph(doc: Document): Graph {
12         return Graph.build {
13             val vertices = hashMapOf<String, Graph.Companion.GraphBuilder.
14                 ↳ VertexBuilder>()
15
16             var res = doc.getElementsByTagName("element")
17             points@ for (i in 0..(res.length - 1)) {
18                 val node = res.item(i) as Element
19                 if (node.getAttribute("type") == "point") {
20                     val name = node.getAttribute("label")
21                     val children = node.getElementsByTagName("coords")
22                     for (j in 0..(children.length - 1)) {
23                         val subNode = children.item(j) as Element
24                         val x = subNode.getAttribute("x").toDouble()
25                         val y = subNode.getAttribute("y").toDouble()
26                         vertices[name] = it.vertex(name, x, y)
27                         continue@points
28                     }
29                 }
30             }
31             res = doc.getElementsByTagName("command")
32             fragments@ for (i in 0..(res.length - 1)) {
33                 val node = res.item(i) as Element
34                 if (node.getAttribute("name") == "Segment") {
35                     val children = node.getElementsByTagName("input")
36                     for (j in 0..(children.length - 1)) {
37                         val subNode = children.item(j) as Element
38                         val a0 = vertices[subNode.getAttribute("a0")]
39                         val a1 = vertices[subNode.getAttribute("a1")]
40                         if (a0 != null && a1 != null)
41                             a0 - a1
42                         continue@fragments
43                     }
44                 }
45             }
46         }
47
48     while (entries.hasMoreElements()) {
49         val entry = entries.nextElement()
50         if (entry.name == "geogebra.xml") {
51             val stream = zipFile.getInputStream(entry)
52             val dbFactory = DocumentBuilderFactory.newInstance()
53             val dBuilder = dbFactory.newDocumentBuilder()
54             val doc = dBuilder.parse(stream)
55             return parseGraph(doc)

```

```

56     }
57 }
58
59 return Graph.build { }
60 }

```

Листинг 6 — LoadGraph.kt

```

1  import java.util.*
2
3  class PriorityQueue<T>(vararg ts: T) {
4
5      private val mp = mutableListOf<Double>()
6      private val mt = mutableListOf<T>()
7
8      init {
9          ts.forEach {
10             push(it, 0.0)
11         }
12     }
13     fun push(t: T, priority: Double) {
14         synchronized(this) {
15             mp.add(priority)
16             mt.add(t)
17         }
18     }
19
20     fun isEmpty() = mp.isEmpty()
21
22     fun isEmpty() = !isEmpty()
23
24     fun pop(): T {
25         if (isEmpty())
26             throw EmptyStackException()
27         synchronized(this) {
28             var max = mp[0]
29             var index = 0
30             var i = 0
31             do {
32                 if (mp[i] < max)
33                     index = i
34             } while (++i < mp.size)
35             mp.removeAt(index)
36             val result = mt[index]
37             mt.removeAt(index)
38             return result
39         }
40     }
41 }

```

Листинг 7 — PriorityQueue.kt