МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

ОТЧЕТ

по лабораторной работе №3

по дисциплине: «Качество и тестирование программного обеспечения»

Выполнил: Евдокимов Н.А.

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71-ПГ

Проверили: Конюхова О.В., Ужаринский А.Ю.

Отметка о зачете:

Дата: «____» _____ 2020 г.

Орёл, 2020

Тема: Функциональное тестирование программного обеспечения.

Задание:

Берём алгоритм из файла: Лабораторная работа 2 Функциональное тестирование. Пишем алгоритм для решения задачи. Для полученной программы по методу причин и следствий строим причинно-следственный граф. На основе графа строим таблицу с комбинациями причин и следствий. Для каждого столбца таблицы пишем тест.

Вариант 4:

Разработать программу определения вида четырехугольника, заданного координатами вершин на плоскости: квадрат, прямоугольник, параллелограмм, ромб, равнобедренная трапеция, прямоугольная трапеция, трапеция общего вида, четырехугольник общего вида.

Исходный код:

```
data class Point(val x: Int, val y: Int)

data class Line(
    val point1: Point,
    val point2: Point
) {
    constructor(points: Pair<Point, Point>) : this(points.first, points.second)

    val A = point1.y - point2.y
    val B = point2.x - point1.x
    val C = point1.x * point2.y + point2.x * point1.y

    fun isPerpendicularTo(line: Line): Boolean {
        return A * line.A + B * line.B == 0
    }

    fun slope(): Double {
        return (point2.y - point1.y).toDouble() / (point2.x - point1.x)
    }

    fun isParallelTo(line: Line): Boolean {
```

```
        val slope1 = slope()
        val slope2 = line.slope()
        val inf = Double.POSITIVE_INFINITY

        return abs(slope1 - slope2) < 0.00000001
                || (abs(slope1) == inf && abs(slope2) == inf)
    }

    fun length(): Double {
        return sqrt(
            (point1.x - point2.x).toDouble().pow(2)
                + (point1.y - point2.y).toDouble().pow(2)
        )
    }

    override fun equals(other: Any?): Boolean {
        if (other is Line) {
            return (other.point1 == point1 && other.point2 == point2)
                    || (other.point1 == point2 && other.point2 == point1)
        }
        return super.equals(other)
    }

    override fun hashCode(): Int {
        var result = point1.hashCode()
        result = 31 * result + point2.hashCode()
        return result
    }
}

data class Quadrangle(
    val point1: Point,
    val point2: Point,
    val point3: Point,
    val point4: Point
) {
    fun lines(): List<Line> {
        return listOf(
            Line(point1 to point2),
            Line(point2 to point3),
```

```
            Line(point3 to point4),
            Line(point4 to point1)
        )
    }

    fun straightAnglesCount(): Int {
        val lines = lines()
        var perpendicularLinesCount = 0
        for (i in 0 until 3) {
            for (j in 3 downTo i + 1) {
                if (lines[i].isPerpendicularTo(lines[j])) {
                    perpendicularLinesCount++
                }
            }
        }

        return perpendicularLinesCount
    }

    fun parallelLinesCount(): Int {
        val lines = lines()
        var parallelLinesCount = 0
        for (i in 0 until 3) {
            for (j in 3 downTo i + 1) {
                if (lines[i].isParallelTo(lines[j])) {
                    parallelLinesCount += 2
                }
            }
        }

        return parallelLinesCount
    }
}

enum class QuadrangleTypes(russianName: String) {
    SQUARE("Квадрат"),
    RECTANGLE("Прямоугольник"),
    PARALLELOGRAM("Параллелограмм"),
    RHOMBUS("Ромб"),
    ISOSCELES_TRAPEZOID("Равнобедренная трапеция"),
```

```
    RECTANGULAR_TRAPEZOID("Прямоугольная трапеция"),
    GENERAL_TRAPEZOID("Трапеция общего вида"),
    GENERAL_QUADRANGLE("Четырехугольник общего вида")
}

class QuadrangleTypeFinder {
    companion object {

        fun determineQuadrangleType(quad: Quadrangle): QuadrangleTypes? {
            val lines = quad.lines()
            val linePair1 = lines[0] to lines[2]
            val linePair2 = lines[1] to lines[3]

            return if (!linePair1.first.isParallelTo(linePair1.second)
                && !linePair2.first.isParallelTo(linePair2.second)
            ) {
                QuadrangleTypes.GENERAL_QUADRANGLE
            } else if ( // Трапеция
                linePair1.first.isParallelTo(linePair1.second)
                && !linePair2.first.isParallelTo(linePair2.second)
                || linePair2.first.isParallelTo(linePair2.second)
                && !linePair1 .first.isParallelTo(linePair1.second)
            ) {
                if (linePair1.first.length() == linePair1.second.length()
                    || linePair2.first.length() == linePair2.second.length()
                ) {
                    QuadrangleTypes.ISOSCELES_TRAPEZOID
                } else if (quad.straightAnglesCount() == 2) {
                    QuadrangleTypes.RECTANGULAR_TRAPEZOID
                } else {
                    QuadrangleTypes.GENERAL_TRAPEZOID
                }
            } else if (lines.all { it.length() == lines[0].length() }) {
                if (quad.straightAnglesCount() == 4) {
                    QuadrangleTypes.SQUARE
                } else {
                    QuadrangleTypes.RHOMBUS
                }
            } else if (quad.straightAnglesCount() == 4) {
                QuadrangleTypes.RECTANGLE
```

```
            } else if (linePair1.first.isParallelTo(linePair1.second)
                && linePair2.first.isParallelTo(linePair2.second)) {
                QuadrangleTypes.PARALLELOGRAM
            } else {
                null
            }
        }
    }
}
```

Причинно-следственный граф:

1. Первая пара отрезков параллельна
2. Вторая пара линий параллельна
3. Длины первой пары отрезков равны
4. Длины второй пары отрезков равны
5. Количество прямых углов == 2
6. Количество прямых углов == 4

101. Четырехугольник общего вида
102. Равнобедренная трапеция
103. Прямоугольная трапеция
104. Трапеция общего вида
105. Квадрат
106. Ромб
107. Прямоугольник
108. Параллелограмм

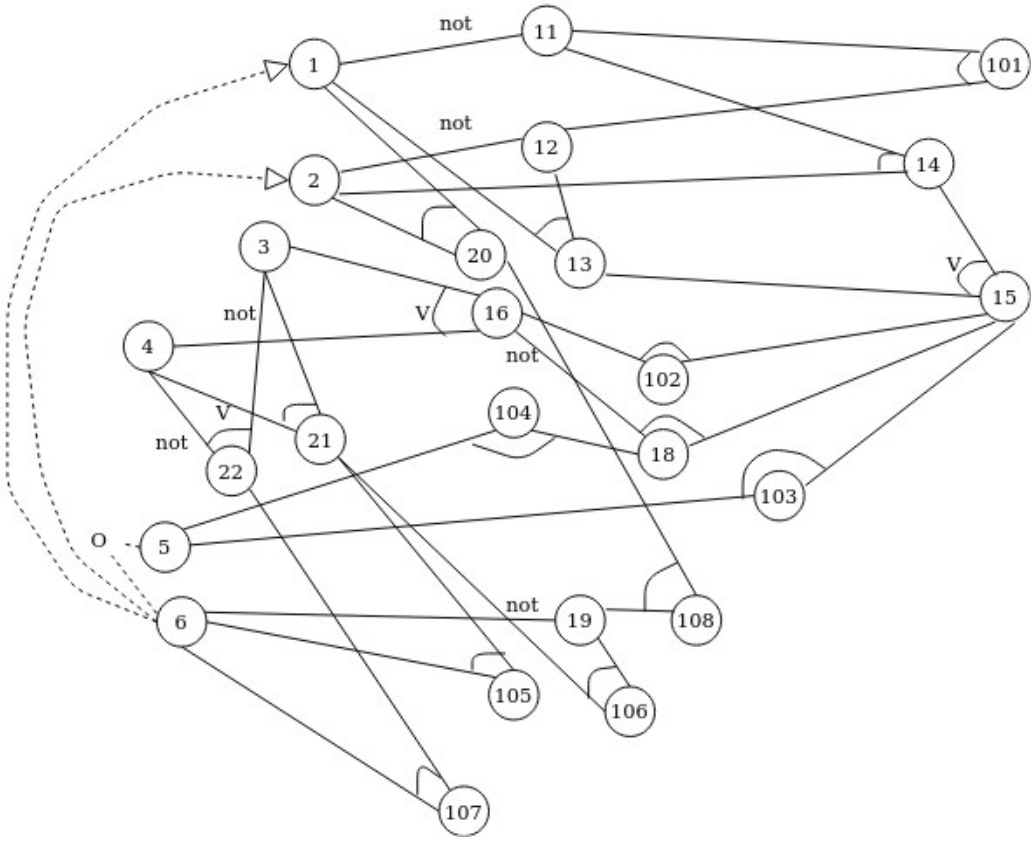

Таблица:

Номер столбца		1	2	3	4	5	6	7	8	9	10	11	12
Причины	1	0	0	1	0	1	0	1	-	-	-	-	1
	2	0	1	0	1	0	1	0	-	-	-	-	1
	3	-	1	0	-	-	0	0	0	0	0	1	-
	4	-	0	1	-	-	0	0	1	1	1	0	-
	5	-	-	-	1	1	0	0	-	-	-	-	-
	6	-	-	-	-	-	-	-	1	0	1	1	0
Вторичные причины	11	1	1	0	1	0	1	0	-	-	-	-	-
	12	1	0	1	0	1	0	1	-	-	-	-	-
	13	-	0	1	0	1	0	1	-	-	-	-	-
	14	-	1	0	1	0	1	0	-	-	-	-	-
	15	-	1	1	1	1	1	1	-	-	-	-	-
	16	-	1	1	-	-	1	1	-	-	-	-	-
	17	-	-	-	-	-	-	-	-	-	-	-	-
	18	-	-	-	-	-	1	1	-	-	-	-	-
	19	-	-	-	-	-	-	-	-	1	-	-	1
	20	-	-	-	-	-	-	-	-	-	-	-	1
	21	-	-	-	-	-	-	-	1	1	1	1	-
Следствия	101	1	0	0	0	0	0	0	0	0	0	0	0
	102	0	1	1	0	0	0	0	0	0	0	0	0
	103	0	0	0	1	1	0	0	0	0	0	0	0
	104	0	0	0	0	0	1	1	0	0	0	0	0
	105	0	0	0	0	0	0	0	1	0	0	0	0
	106	0	0	0	0	0	0	0	0	1	0	0	0
	107	0	0	0	0	0	0	0	0	0	1	1	0
	108	0	0	0	0	0	0	0	0	0	0	0	1

Тесты:

```
import org.junit.jupiter.api.Test
import kotlin.test.assertEquals

class QuadrangleTypeFinderProperTest {
    @Test
    fun `Both pairs are not parallel`() {
        val generalQuadrangle = Quadrangle(
            Point(0, 0), Point(5, 2), Point(9, 3), Point(1, -2)
        )
        val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(generalQuadrangle)

        assertEquals(QuadrangleTypes.GENERAL_QUADRANGLE, determinedType)
    }
```

```
        @Test
        fun `Second pair is parallel, first pair lengths are equal`() {
            val isoscelesTrapezoid = Quadrangle(
                Point(0, 0), Point(1, 1), Point(2, 1), Point(3, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(isoscelesTrapezoid)

            assertEquals(QuadrangleTypes.ISOSCELES_TRAPEZOID, determinedType)
        }

        @Test
        fun `First pair is parallel, second pair lengths are equal`() {
            val isoscelesTrapezoid = Quadrangle(
                Point(0, 0), Point(0, 3), Point(1, 2), Point(1, 1)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(isoscelesTrapezoid)

            assertEquals(QuadrangleTypes.ISOSCELES_TRAPEZOID, determinedType)
        }

        @Test
        fun `Second pair is parallel, straight angles count = 2`() {
            val rectangularTrapezoid = Quadrangle(
                Point(0, 0), Point(0, 1), Point(1, 1), Point(2, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(rectangularTrapezoid)

            assertEquals(QuadrangleTypes.RECTANGULAR_TRAPEZOID, determinedType)
        }

        @Test
        fun `First pair is parallel, straight angles count = 2`() {
            val rectangularTrapezoid = Quadrangle(
                Point(0, 0), Point(0, 1), Point(1, 2), Point(1, 0)
            )
```

```
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(rectangularTrapezoid)

            assertEquals(QuadrangleTypes.RECTANGULAR_TRAPEZOID, determinedType)
        }

        @Test
        fun `Second pair is parallel, straight angles count != 2`() {
            val generalTrapezoid = Quadrangle(
                Point(0, 0), Point(1, 2), Point(2, 2), Point(6, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(generalTrapezoid)

            assertEquals(QuadrangleTypes.GENERAL_TRAPEZOID, determinedType)
        }

        @Test
        fun `First pair is parallel, straight angles count != 2`() {
            val generalTrapezoid = Quadrangle(
                Point(0, 0), Point(0, 6), Point(2, 2), Point(2, 1)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(generalTrapezoid)

            assertEquals(QuadrangleTypes.GENERAL_TRAPEZOID, determinedType)
        }

        @Test
        fun `Straight angles count = 4, both pairs lengths are equal`() {
            val square = Quadrangle(
                Point(0, 0), Point(0, 1), Point(1, 1), Point(1, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(square)

            assertEquals(QuadrangleTypes.SQUARE, determinedType)
        }

        @Test
```

```
        fun `Straight angles count != 4, both pairs lengths are equal`() {
            val rhombus = Quadrangle(
                Point(1, 1), Point(3, 5), Point(7, 7), Point(5, 3)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(rhombus)

            assertEquals(QuadrangleTypes.RHOMBUS, determinedType)
        }

        @Test
        fun `Straight angles count = 4, first pairs lengths are equal`() {
            val rectangle = Quadrangle(
                Point(0, 0), Point(0, 1), Point(2, 1), Point(2, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(rectangle)

            assertEquals(QuadrangleTypes.RECTANGLE, determinedType)
        }

        @Test
        fun `Straight angles count = 4, second pairs lengths are equal`() {
            val rectangle = Quadrangle(
                Point(0, 0), Point(0, 2), Point(1, 2), Point(1, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(rectangle)

            assertEquals(QuadrangleTypes.RECTANGLE, determinedType)
        }

        @Test
        fun `Straight angles count != 4, both pairs are parallel`() {
            val parallelogram = Quadrangle(
                Point(0, 0), Point(1, 1), Point(3, 1), Point(2, 0)
            )
            val determinedType =
QuadrangleTypeFinder.determineQuadrangleType(parallelogram)
```

```
            assertEquals(QuadrangleTypes.PARALLELOGRAM, determinedType)
        }
    }
```