Звіт першої лабораторної роботи  
з теми «Визначення приналежності точки до опуклого многокутника»  
Точаненка Владислава Володимировича, група ІПС-31

**ОСНОВНІ ПОЛОЖЕННЯ**

Для того щоб **визначити з якої сторони лежить точка C відносно вектору AB** можна використовувати знак третьої координати результату векторного добутку векторів AB і AC. Вона вираховується за такою формулою: . Якщо , то точка лежить зліва від вектору, якщо , то з права від вектору, а якщо 0, то вони лежать на одній прямій. Функція rotation отримує три точки і повертає координату Z векторного добутку векторів AB і AC:

fun rotation(a: Dot, b: Dot, c: Dot): Double

    = ((b.x - a.x) \* (c.y - b.y) - (b.y - a.y) \* (c.x - b.x))

Для того щоб **визначити ти перетинає один відрізок інший** можна визначити чи лежать кінці одного відрізка по різні сторони від другого і навпаки. Треба перевіряти обидва варіанти, бо якщо перевіримо відносно одного відрізку, то насправді ми перевіримо відносно прямої. Функція intersect отримає чотири точки A, B, C і D та повертає true якщо відрізки AB і CD пересікаються, false – якщо не пересікаються.

fun intersect(a: Dot, b: Dot, c: Dot, d: Dot): Boolean =

    (rotation(a, b, c) \* rotation(a, b, d) <= 0)

    && (rotation(c, d, a) \* rotation(c, d, b) < 0)

**ОПИС АЛГОРИТМУ**

**Вхідні дані:** N координат опуклого многокутника і координата точки

**Вихідні дані:** Малюнок многокутника і точки, а також вивід чи належить точка многокутнику чи не належить.

1. Сортуємо вершини многокутника так, щоб вони були впорядковані проти годинникової стрілки, при чому перша вершина буде із найменшою координатою за x.
2. Перевіряємо чи належить крапка півплощині, яка розділяється кутом Pn-1P0P1, якщо Pi – і-та першина многокутника. Якщо вона лежить зліва від цього кута, то повертається false. Якщо ж ні, ідемо за алгоритмом далі.

val len = vertexes.size

if ((rotation(vertexes[0], vertexes[1], dot) < 0)

    || (rotation(vertexes[0], vertexes[len - 1], dot) > 0)

)

    return false

1. Обираємо ліву вершину як першу, а праву як останню. Далі як бінарним пошуком шукаємо у якому секторі кута лежить точка. Для цього обираємо середню вершину Pm між лівою Pl і правою Pr вершинами многокутника що лежать на сторонах поточного кута. Визначаємо з якої сторони лежить крапка від вектору P0Pm. Якщо знаходиться зліва, то r = m, якщо з права, то l = m.
2. Повторюємо крок 3, поки не буде різниця r – l дорівнювати 1.

var l = 1

var r = len - 1

var m: Int

while (r - l > 1) {

    m = l + (r - l) / 2

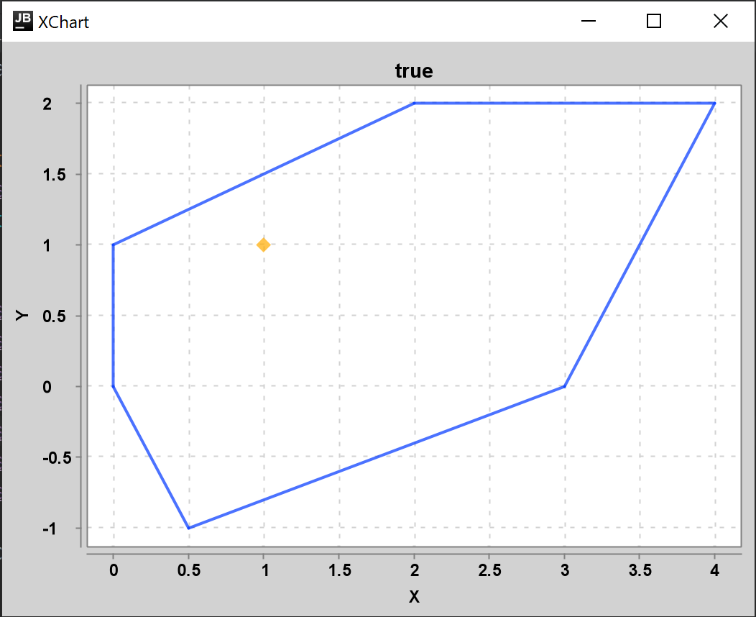
    if (rotation(vertexes[0], vertexes[m], dot) < 0) r = m

    else l = m

}

1. Тепер ми визначили сектор кута, у якому лежить точка. Тепер треба визначити чи лежить точка всередині многокутника, чи за його межами. Для цього використаємо функцію intersect. Результат її виконання буде протилежний до потрібного результату. Тобто якщо функція intersect поверне true, то відрізки пересікаються і точка лежить за межами многокутника, тобто результат буде false, і навпаки.

return !intersect(vertexes[0], dot, vertexes[l], vertexes[r])



**Dot.kt**

data class Dot (

    var x: Double,

    var y: Double

)

**Main.kt**

import kotlin.math.atan2

class Main {

    companion object {

        var vertexes = mutableListOf<Dot>()

        var dot: Dot

        init {

            vertexes.add(Dot(0.0, 1.0))

            vertexes.add(Dot(0.0, 0.0))

            vertexes.add(Dot(2.0, 2.0))

            vertexes.add(Dot(4.0, 2.0))

            vertexes.add(Dot(3.0, 0.0))

            vertexes.add(Dot(0.5, -1.0))

            vertexes.sortBy { -atan2(-it.x, -it.y) }

//            dot = Dot(-0.1, 0.9)

            dot = Dot(1.0, 1.0)

        }

        fun rotation(a: Dot, b: Dot, c: Dot): Double = ((b.x - a.x) \* (c.y - b.y) - (b.y - a.y) \* (c.x - b.x))

        fun intersect(a: Dot, b: Dot, c: Dot, d: Dot): Boolean =

            (rotation(a, b, c) \* rotation(a, b, d) <= 0)

                    && (rotation(c, d, a) \* rotation(c, d, b) < 0)

        fun checkIfInside(): Boolean {

            val len = vertexes.size

            if ((rotation(vertexes[0], vertexes[1], dot) < 0)

                || (rotation(vertexes[0], vertexes[len - 1], dot) > 0)

            )

                return false

            var l = 1

            var r = len - 1

            var m: Int

            while (r - l > 1) {

                m = l + (r - l) / 2

                if (rotation(vertexes[0], vertexes[m], dot) < 0) r = m

                else l = m

            }

            return !intersect(vertexes[0], dot, vertexes[l], vertexes[r])

        }

        @JvmStatic

        fun main(args: Array<String>) {

            Visualizer.draw(vertexes, dot)

            Visualizer.setTitle(checkIfInside().toString())

        }

    }

}

**Visualizer.kt**

import org.knowm.xchart.QuickChart

import org.knowm.xchart.SwingWrapper

import org.knowm.xchart.XYChart

class Visualizer {

    companion object {

        lateinit var chart: XYChart

        @JvmStatic

        fun draw(dots: List<Dot>, dot: Dot) {

            val x = mutableListOf<Double>()

            val y = mutableListOf<Double>()

            for (singleDot in dots) {

                x.add(singleDot.x)

                y.add(singleDot.y)

            }

            x.add(dots[0].x)

            y.add(dots[0].y)

            chart = QuickChart.getChart("Sample Chart", "X", "Y", "y(x)", x, y)

            chart.addSeries("DOT", doubleArrayOf(dot.x), doubleArrayOf(dot.y))

            SwingWrapper(chart).displayChart()

        }

        @JvmStatic

        fun setTitle(string: String) {

            chart.title = string

        }

    }

}