Белорусский государственный университет
информатики и радиоэлектроники
Кафедра ПОИТ

Контрольная работа №1

«Численные алгоритмы»

Выполнил: ст. гр. 613801 Шилов В.А.

Проверил: Скобцов В.Ю.

Минск 2019

1. Транспонировать квадратную матрицу на том же месте

```swift
import Foundation

protocol ITransposition {
    func transposite()
}

final class Transposition: ITransposition {
    private var originalMatrix: [[Int]]
    private var finalMatrix: [[Int]]

    init(numberOfLines: Int, numberOfColumn: Int) {
        originalMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)
        finalMatrix = Array(repeating: Array(repeating: 0, count: numberOfLines), count: numberOfColumn)

        originalMatrix = originalMatrix.map { line -> [Int] in
            line.map({ element -> Int in                                    (6 times)
                return Int(arc4random_uniform(10))                          (54 times)
            })
        }
    }

    func transposite() {
        for (lineIndex, line) in originalMatrix.enumerated() {
            for columnIndex in 0..<line.count {
                finalMatrix[columnIndex][lineIndex] = originalMatrix[lineIndex][columnIndex]   (54 times)
            }
        }

        printResult(resultName: "Original matrix", result: originalMatrix)  Transposition
        printResult(resultName: "Tranposited matrix", result: finalMatrix)  Transposition
    }

    private func printResult(resultName: String, result: [[Int]]) {
        print(resultName)                                                   (2 times)
        result.forEach { line in                                            (2 times)
            print(line)                                                     (15 times)
        }
        print()                                                             (2 times)
    }
}

let transposition: ITransposition = Transposition(numberOfLines: 6, numberOfColumn: 9)   Transposition
transposition.transposite()
```

```
Original matrix
[8, 3, 6, 6, 3, 6, 2]
[7, 3, 3, 4, 6, 2, 7]
[1, 3, 7, 1, 8, 8, 3]
[2, 3, 7, 9, 5, 2, 2]
[0, 9, 6, 4, 0, 0, 9]

Tranposited matrix
[8, 7, 1, 2, 0]
[3, 3, 3, 3, 9]
[6, 3, 7, 7, 6]
[6, 4, 1, 9, 4]
[3, 6, 8, 5, 0]
[6, 2, 8, 2, 0]
[2, 7, 3, 2, 9]
```

```
Original matrix
[7, 4, 5, 0, 9, 7, 0, 9, 9]
[3, 3, 9, 2, 8, 6, 2, 4, 0]
[3, 1, 0, 8, 1, 6, 0, 9, 2]
[2, 4, 2, 4, 0, 0, 4, 7, 4]
[5, 6, 5, 9, 4, 2, 6, 1, 0]
[4, 3, 5, 4, 3, 4, 7, 4, 3]

Tranposited matrix
[7, 3, 3, 2, 5, 4]
[4, 3, 1, 4, 6, 3]
[5, 9, 0, 2, 5, 5]
[0, 2, 8, 4, 9, 4]
[9, 8, 1, 0, 4, 3]
[7, 6, 6, 0, 2, 4]
[0, 2, 0, 4, 6, 7]
[9, 4, 9, 7, 1, 4]
[9, 0, 2, 4, 0, 3]
```

```
Original matrix
[7, 3, 4, 9, 3, 3, 8]
[6, 5, 5, 6, 2, 1, 8]
[5, 5, 6, 7, 9, 2, 8]
[0, 0, 2, 5, 3, 7, 0]
[2, 3, 3, 0, 7, 4, 0]

Tranposited matrix
[7, 6, 5, 0, 2]
[3, 5, 5, 0, 3]
[4, 5, 6, 2, 3]
[9, 6, 7, 5, 0]
[3, 2, 9, 3, 7]
[3, 1, 2, 7, 4]
[8, 8, 8, 0, 0]
```

2. Сложить две матрицы вещественных чисел.
Для начала стоит сказать, что я добавил несколько интерфейсов для того, чтобы в будущем было более удобно работать с матрицами:

Вывод результата:

```swift
protocol Resulter {
    func printResult(resultName: String, result: [[Int]])
}

extension Resulter {
    func printResult(resultName: String, result: [[Int]]) {
        print(resultName)
        result.forEach { line in
            print(line)
        }
        print()
    }
}
```

Добавление значений в матрицы:

```swift
protocol IPopulator {
    associatedtype TypeToPopulate

    func populate(_ type: inout TypeToPopulate)
}

protocol IMatrixPopulator: IPopulator where TypeToPopulate == [[Int]] {
    func populate(_ type: inout TypeToPopulate)
}

final class MatrixPopulator: IMatrixPopulator {
    func populate(_ type: inout TypeToPopulate) {
        type = type.map { line -> [Int] in
            line.map({ element -> Int in
                return Int(arc4random_uniform(10))
            })
        }
    }
}
```

Само решение:

```swift
protocol IMatrixAddition {
    func add()
    func showResult()
}

final class MatrixAddition: IMatrixAddition, Resulter {
    // MARK: - Helpers
    private let matrixPopulator = MatrixPopulator()

    // MARK: - Properties
    private var firstMatrix: [[Int]]
    private var secondMatrix: [[Int]]
    private var sumMatrix: [[Int]]

    init(numberOfLines: Int, numberOfColumn: Int) {
        firstMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)
        secondMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)
        sumMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)

        matrixPopulator.populate(&firstMatrix)
        matrixPopulator.populate(&secondMatrix)

        printResult(resultName: "first", result: firstMatrix)
        printResult(resultName: "second", result: secondMatrix)
    }

    func add() {
        for (lineIndex, line) in firstMatrix.enumerated() {
            for columnIndex in 0..<line.count {
                sumMatrix[lineIndex][columnIndex] = firstMatrix[lineIndex][columnIndex] + secondMatrix[lineIndex][columnIndex]
            }
        }
    }

    func showResult() {
        printResult(resultName: "sum", result: sumMatrix)
    }
}

let matrixAddition: IMatrixAddition = MatrixAddition(numberOfLines: 5, numberOfColumn: 5)
matrixAddition.add()
matrixAddition.showResult()
```

(25 times)

MatrixAddition

MatrixAddition

```
first
[1, 1, 3, 6, 0]
[2, 0, 7, 9, 1]
[3, 0, 4, 7, 1]
[9, 7, 1, 8, 2]
[1, 0, 0, 1, 8]

second
[2, 5, 5, 7, 0]
[9, 4, 9, 3, 5]
[5, 7, 8, 4, 7]
[2, 3, 6, 3, 3]
[6, 8, 4, 0, 9]

sum
[3, 6, 8, 13, 0]
[11, 4, 16, 12, 6]
[8, 7, 12, 11, 8]
[11, 10, 7, 11, 5]
[7, 8, 4, 1, 17]
```

```
first
[8, 7, 8, 8, 0]
[8, 9, 2, 9, 4]
[8, 1, 2, 4, 8]
[3, 8, 8, 0, 9]
[3, 4, 3, 8, 5]

second
[7, 1, 6, 3, 6]
[8, 2, 7, 2, 7]
[4, 7, 9, 3, 2]
[8, 5, 9, 9, 3]
[9, 5, 0, 2, 3]

sum
[15, 8, 14, 11, 6]
[16, 11, 9, 11, 11]
[12, 8, 11, 7, 10]
[11, 13, 17, 9, 12]
[12, 9, 3, 10, 8]
```

```
first
[3, 5, 6, 1, 9]
[8, 4, 8, 1, 5]
[6, 2, 1, 6, 5]
[4, 2, 0, 5, 8]
[7, 7, 9, 6, 7]

second
[9, 6, 0, 0, 8]
[7, 2, 2, 7, 0]
[3, 6, 7, 7, 2]
[6, 6, 4, 1, 2]
[0, 2, 9, 8, 5]

sum
[12, 11, 6, 1, 17]
[15, 6, 10, 8, 5]
[9, 8, 8, 13, 7]
[10, 8, 4, 6, 10]
[7, 9, 18, 14, 12]
```

3. Перемножить две матрицы вещественных чисел.

```swift
final class MatrixMultiplication: IMatrixMultiplication, Resulter {
    // MARK: - Helpers
    private let matrixPopulator = MatrixPopulator()

    // MARK: - Properties
    private var firstMatrix: [[Int]]
    private var secondMatrix: [[Int]]
    private var resultMatrix: [[Int]]

    private let numberOfLines: Int
    private let numberOfColumn: Int
    private let numberOfFirstMatrixColumn: Int

    init(numberOfFirstMatrixLines: Int, numberOfFirstMatrixColumn: Int, numberOfSecondMatrixColumn: Int) {
        firstMatrix = Array(repeating: Array(repeating: 0, count: numberOfFirstMatrixColumn), count: numberOfFirstMatrixLines)
        secondMatrix = Array(repeating: Array(repeating: 0, count: numberOfSecondMatrixColumn), count: numberOfFirstMatrixColumn)

        self.numberOfLines = numberOfFirstMatrixLines
        self.numberOfColumn = numberOfSecondMatrixColumn
        self.numberOfFirstMatrixColumn = numberOfFirstMatrixColumn
        resultMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)

        matrixPopulator.populate(&firstMatrix)
        matrixPopulator.populate(&secondMatrix)

        printResult(resultName: "first", result: firstMatrix)
        printResult(resultName: "second", result: secondMatrix)
    }


    func multiply() {
        for lineIndex in 0..<numberOfLines {
            for columnIndex in 0..<numberOfColumn {
                for index in 0..<numberOfFirstMatrixColumn {
                    resultMatrix[lineIndex][columnIndex] += firstMatrix[lineIndex][index] * secondMatrix[index][columnIndex]
                }
            }
        }
    }

    func showResult() {
        printResult(resultName: "result", result: resultMatrix)
    }
```

Related Items
MatrixMultiplication

```
first
[1, 4, 1, 6, 1, 9, 0]
[0, 6, 8, 4, 9, 6, 9]
[9, 9, 6, 5, 1, 2, 2]
[3, 9, 1, 9, 6, 5, 1]

second
[7, 1, 9, 9, 2]
[6, 8, 2, 7, 6]
[7, 6, 5, 4, 6]
[2, 8, 9, 6, 9]
[6, 7, 1, 5, 5]
[8, 5, 6, 9, 2]
[0, 3, 6, 8, 8]

result
[128, 139, 131, 163, 109]
[202, 248, 187, 269, 249]
[191, 180, 199, 237, 178]
[176, 223, 173, 231, 195]
```

```
first
[8, 1]

second
[1, 9, 8]
[1, 7, 0]

result
[9, 79]
```

```
first
[7, 6, 3, 7]
[7, 1, 7, 7]
[2, 3, 0, 6]
[3, 4, 9, 6]
[7, 5, 4, 5]

second
[5, 6, 8]
[5, 5, 5]
[3, 0, 2]
[4, 7, 6]

result
[102, 121, 134]
[89, 96, 117]
[49, 69, 67]
[86, 80, 98]
[92, 102, 119]
```

4. Перемножить две матрицы вещественных чисел используя алгоритм Винограда.

При первоначальном проектировании была допущена ошибка, поэтому я вынес базовый функционал для всех множителей матриц в базовый класс:

```swift
class BaseMatrixMultiplication: IMatrixMultiplication, Resulter {
    // MARK: - Helpers
    private let matrixPopulator = MatrixPopulator()

    // MARK: - Properties
    var firstMatrix: [[Int]]
    var secondMatrix: [[Int]]
    var resultMatrix: [[Int]]

    let numberOfLines: Int
    let numberOfColumn: Int
    let numberOfFirstMatrixColumn: Int

    init(numberOfFirstMatrixLines: Int, numberOfFirstMatrixColumn: Int, numberOfSecondMatrixColumn: Int) {
        firstMatrix = Array(repeating: Array(repeating: 0, count: numberOfFirstMatrixColumn), count: numberOfFirstMatrixLines)
        secondMatrix = Array(repeating: Array(repeating: 0, count: numberOfSecondMatrixColumn), count: numberOfFirstMatrixColumn)

        self.numberOfLines = numberOfFirstMatrixLines
        self.numberOfColumn = numberOfSecondMatrixColumn
        self.numberOfFirstMatrixColumn = numberOfFirstMatrixColumn
        resultMatrix = Array(repeating: Array(repeating: 0, count: numberOfColumn), count: numberOfLines)

        matrixPopulator.populate(&firstMatrix)
        matrixPopulator.populate(&secondMatrix)

        printResult(resultName: "first", result: firstMatrix)
        printResult(resultName: "second", result: secondMatrix)
    }

    func multiply() { }
    func showResult() { }
}
```

Общение же с этим классами продолжается исключительно через интерфейс, поэтому все переменный класса недоступны извне:

```swift
let vinogradovMultiplication: IMatrixMultiplication = VinogradovMultiplication(numberOfFirstMatrixLines: 1, numberOfFirstMatrixColumn: 2,
    numberOfSecondMatrixColumn: 3)
vinogradovMultiplication.
vinogradovMultip
```

VinogradovMultiplication

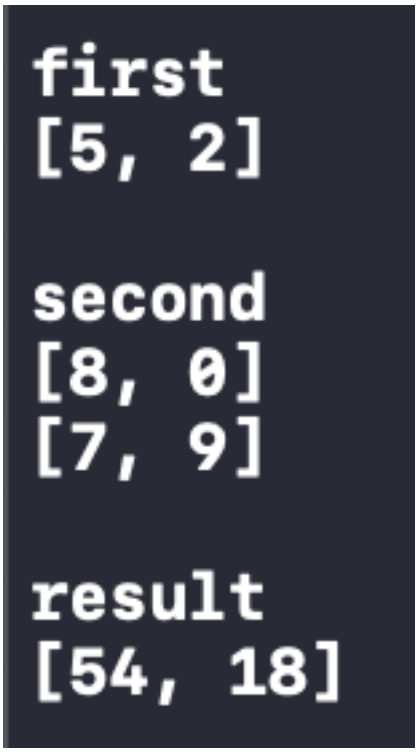
Void multiply()
Void showResult()

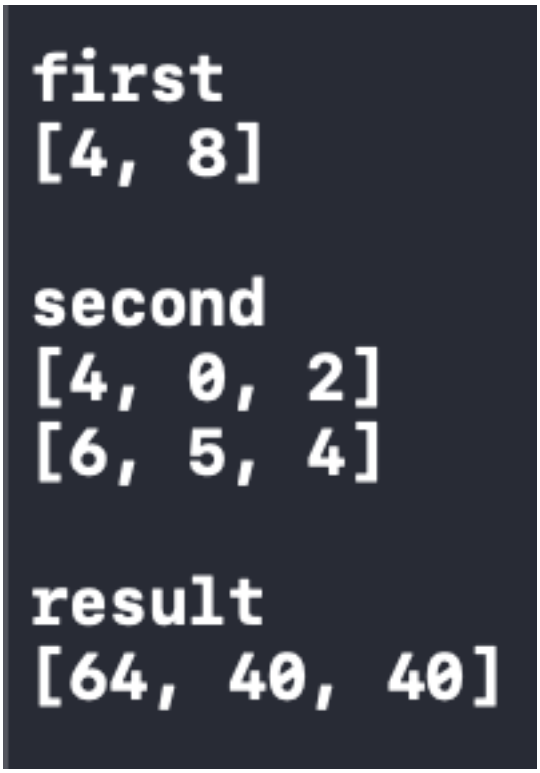
```swift
final class VinogradovMultiplication: BaseMatrixMultiplication {
    var rowFactor: [Int]
    var columnFactor: [Int]

    override init(numberOfFirstMatrixLines: Int, numberOfFirstMatrixColumn: Int, numberOfSecondMatrixColumn: Int) {
        rowFactor = [Int].init(repeating: 0, count: numberOfSecondMatrixColumn)
        columnFactor = [Int].init(repeating: 0, count: numberOfSecondMatrixColumn)

        super.init(numberOfFirstMatrixLines: numberOfFirstMatrixLines, numberOfFirstMatrixColumn: numberOfFirstMatrixColumn, numberOfSecondMatrixColumn:
            numberOfSecondMatrixColumn)
    }

    override func multiply() {
        let d = numberOfFirstMatrixColumn / 2                                                                                    // 1

        // вычисление rowFactors
        for lineIndex in 0..<numberOfLines {
            rowFactor[lineIndex] = firstMatrix[lineIndex][0] * firstMatrix[lineIndex][1]                                         // 10
            for index in 1..<d {
                rowFactor[lineIndex] += firstMatrix[lineIndex][2 * index - 1] * firstMatrix[lineIndex][2 * index]
            }
        }

        // вычисление columnFactors
        for columnIndex in 0..<numberOfColumn {
            columnFactor[columnIndex] = secondMatrix[0][columnIndex] * secondMatrix[1][columnIndex]                              // (2 times)
            for index in 1..<d {
                columnFactor[columnIndex] += secondMatrix[2 * index - 1][columnIndex] * secondMatrix[2 * index][columnIndex]
            }
        }

        // вычисление матрицы R
        for lineIndex in 0..<numberOfLines {
            for columnIndex in 0..<numberOfColumn {
                resultMatrix[lineIndex][columnIndex] = -rowFactor[lineIndex] - columnFactor[columnIndex]                          // (2 times)
                for index in 0..<d {
                    let firstComponent = firstMatrix[lineIndex][2 * index + 1] + secondMatrix[2 * index][columnIndex]             // (2 times)
                    let secondComponent = firstMatrix[lineIndex][2 * index] + secondMatrix[2 * index + 1][columnIndex]           // (2 times)
                    resultMatrix[lineIndex][columnIndex] += firstComponent * secondComponent
                }
            }
        }

```

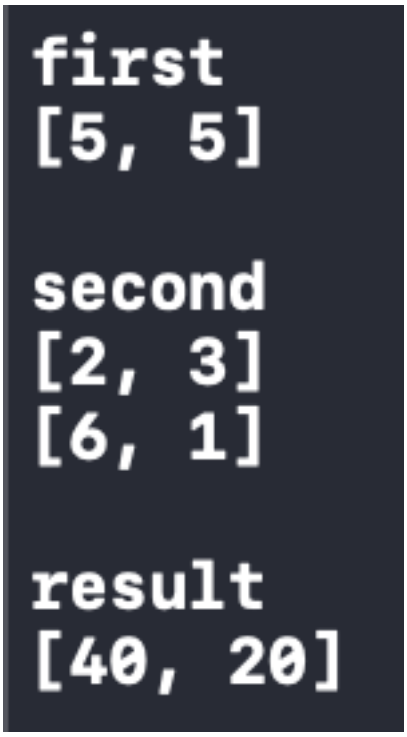
```swift
        // прибавление членов в случае нечетной общей размерности
        if numberOfFirstMatrixColumn % 2 == 1 {
            for lineIndex in 0..<numberOfLines {
                for columnIndex in 0..<numberOfColumn {
                    resultMatrix[lineIndex][columnIndex] += firstMatrix[lineIndex][numberOfLines - 1] * secondMatrix[numberOfLines - 1][columnIndex]
                }
            }
        }
    }

    override func showResult() {
        printResult(resultName: "result", result: resultMatrix)
    }
}

let vinogradovMultiplication: IMatrixMultiplication = VinogradovMultiplication(numberOfFirstMatrixLines: 1, numberOfFirstMatrixColumn: 2,
    numberOfSecondMatrixColumn: 2)
vinogradovMultiplication.multiply()
vinogradovMultiplication.showResult()
```

VinogradovMultiplication

VinogradovMultiplication







5. Перемножить две матрицы вещественных чисел используя алгоритм Штрассена.

```swift
final class ShtrassenMultiplication: BaseMatrixMultiplication {
    override func multiply() {
        firstMatrix = equalize(matrix: firstMatrix)                                    ShtrassenMultiplication
        secondMatrix = equalize(matrix: secondMatrix)                                  ShtrassenMultiplication

        firstMatrix[0][0] = 1                                                          1
        firstMatrix[0][1] = 9                                                          9
        firstMatrix[1][0] = 7                                                          7
        firstMatrix[1][1] = 3                                                          3

        secondMatrix[0][0] = 5                                                         5
        secondMatrix[0][1] = 2                                                         2
        secondMatrix[1][0] = 4                                                         4
        secondMatrix[1][1] = 11                                                        11

        printResult(resultName: "first equalized", result: firstMatrix)                ShtrassenMultiplication
        printResult(resultName: "second equalized", result: secondMatrix)              ShtrassenMultiplication

        asd(firstMatrixCopy: firstMatrix, secondMatrixCopy: secondMatrix)              ShtrassenMultiplication
    }

    override func showResult() {
        printResult(resultName: "result", result: resultMatrix)                        ShtrassenMultiplication
    }

    private func asd(firstMatrixCopy: [[Int]], secondMatrixCopy: [[Int]]) {
        //for index in stride(from: 0, to: firstMatrixCopy.count, by: firstMatrixCopy.count / 2) {
        if firstMatrixCopy.count > 2 {
            let firstDivideResult = divide(matrix: firstMatrixCopy)
            let f1 = firstDivideResult.m1
            let f2 = firstDivideResult.m2
            let f3 = firstDivideResult.m3
            let f4 = firstDivideResult.m4

            let secondDivideResult = divide(matrix: secondMatrixCopy)
            let s1 = secondDivideResult.m1
            let s2 = secondDivideResult.m2
            let s3 = secondDivideResult.m3
            let s4 = secondDivideResult.m4

            asd(firstMatrixCopy: f1, secondMatrixCopy: s1)
            asd(firstMatrixCopy: f2, secondMatrixCopy: s2)
            asd(firstMatrixCopy: f3, secondMatrixCopy: s3)
            asd(firstMatrixCopy: f4, secondMatrixCopy: s4)
        }
        else {
            let firstMatrixEqulized = equalize(matrix: firstMatrixCopy)                [[1, 9], [7, 3]]
            let secondMatrixEqulized = equalize(matrix: secondMatrixCopy)              [[5, 2], [4, 11]]

            shtrassenMultiplication(firstMatrix: firstMatrixEqulized, secondMatrix: secondMatrixEqulized)   ShtrassenMultiplication
        }
        //}
    }

    private func equalize(matrix: [[Int]]) -> [[Int]] {
        var matrixCopy = matrix                                                        (4 times)
        if matrixCopy[0].count % 2 != 0 {
            matrixCopy = matrixCopy.map { line -> [Int] in
                var newLine = line
                newLine.append(0)
                return newLine
            }
        }
        if matrixCopy.count % 2 != 0 {
            matrixCopy.append([Int].init(repeating: 0, count: matrixCopy[0].count))
        }

        return matrixCopy                                                              (4 times)
    }
    private func divide(matrix: [[Int]]) -> (m1: [[Int]], m2: [[Int]], m3: [[Int]], m4: [[Int]]) {
        var matrixCopy = matrix
        matrixCopy = equalize(matrix: matrix)

        let divideFactor = (matrix.count + 1) / 2
        var m1 = Array(repeating: Array(repeating: 0, count: divideFactor), count: divideFactor)
        var m2 = Array(repeating: Array(repeating: 0, count: divideFactor), count: divideFactor)
        var m3 = Array(repeating: Array(repeating: 0, count: divideFactor), count: divideFactor)
        var m4 = Array(repeating: Array(repeating: 0, count: divideFactor), count: divideFactor)

        for (lineIndex, line) in matrix.enumerated() {
            for (columnIndex, column) in line.enumerated() {   ⚠ Immutable value 'column' was never used; consider replacing with '_' or removing it
                if lineIndex < divideFactor {
                    if columnIndex < divideFactor {
                        m1[lineIndex][columnIndex] = matrix[lineIndex][columnIndex]
                    }
                    else {
                        m2[lineIndex][columnIndex - divideFactor] = matrix[lineIndex][columnIndex]
                    }
                }
                else {
                    if columnIndex < divideFactor {
                        m3[lineIndex - divideFactor][columnIndex] = matrix[lineIndex][columnIndex]
                    }
                    else {
                        m4[lineIndex - divideFactor][columnIndex - divideFactor] = matrix[lineIndex][columnIndex]
                    }
                }
            }
        }

        printResult(resultName: "piece of that", result: matrixCopy)
        printResult(resultName: "is that", result: m1)
        printResult(resultName: "and", result: m2)
        printResult(resultName: "and", result: m3)
        printResult(resultName: "and", result: m4)

```

```swift
    private func shtrassenMultiplication(firstMatrix: [[Int]], secondMatrix: [[Int]]) {
        let x1 = (firstMatrix[0][0] + firstMatrix[1][1]) * (secondMatrix[0][0] + secondMatrix[1][1])
        let x2 = (firstMatrix[1][0] + firstMatrix[1][1]) * secondMatrix[0][0]
        let x3 = firstMatrix[0][0] * (secondMatrix[0][1] - secondMatrix[1][1])
        let x4 = firstMatrix[1][1] * (secondMatrix[1][0] - secondMatrix[0][0])
        let x5 = (firstMatrix[0][0] + firstMatrix[0][1]) * secondMatrix[1][1]
        let x6 = (firstMatrix[1][0] - firstMatrix[0][0]) * (secondMatrix[0][0] + secondMatrix[0][1])
        let x7 = (firstMatrix[0][1] - firstMatrix[1][1]) * (secondMatrix[1][0] + secondMatrix[1][1])

        var result = Array(repeating: Array(repeating: 0, count: 2), count: 2)

        result[0][0] = x1 + x4 - x5 + x7
        result[0][1] = x3 + x5
        result[1][0] = x2 + x4
        result[1][1] = x1 - x2 + x3 + x6

        printResult(resultName: "shtrassen result", result: result)
    }
}

let shtrassenMultiplication: IMatrixMultiplication = ShtrassenMultiplication(numberOfFirstMatrixLines: 2, numberOfFirstMatrixColumn: 2,
    numberOfSecondMatrixColumn: 2)
shtrassenMultiplication.multiply()
shtrassenMultiplication.showResult()
```

```
64
50
-9
-3
110
42
90
[[0, 0], [0, 0]]
41
101
47
47
ShtrassenMultiplication
ShtrassenMultiplication
```

Результат для данного в задании примера:

```
first equalized
[1, 9]
[7, 3]

second equalized
[5, 2]
[4, 11]

shtrassen result
[41, 101]
[47, 47]
```

Рандомные значения, сверенные с онлайн калькулятором:

```
first equalized
[6, 8]
[3, 7]

second equalized
[5, 1]
[4, 1]

shtrassen result
[62, 14]
[43, 10]
```

Решение:

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 6 & 8 \\ 3 & 7 \end{pmatrix} \cdot \begin{pmatrix} 5 & 1 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 62 & 14 \\ 43 & 10 \end{pmatrix}$$