Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Факультет информационных технологий и кибербезопасности

Кафедра информационных систем и программирования

**ОТЧЕТ**

по лабораторной работе №8.

Тема работы: «Тестирование с использованием тестовых двойников (Test Doubles)».

Наименование дисциплины: «Тестирование и отладка программного обеспечения».

Подготовил:

Студент группы 23-КБ-ПР1

Булгаков В. В.

Краснодар

2025

**Цель работы**

Цель работы – изучить подход к автоматизации процесса тестирования с использованием тестовых двойников (Test Doubles).

**Задание**

1) Создать класс, реализующий граф/дерево, хранящий информацию о структуре графа, на основе некоторого внутреннего представления указанного в задании из пункта 5.1.

2) В созданном классе реализовать один из методов обработки графа/дерева в соответствии с вариантом задания из пункта 5.2.

3) В качестве источника данных использовать класс, реализующий чтение из файла произвольного формата и возвращающего данные в виде, указанном в варианте задания из пункта 5.1.

4) Реализовать возможность сохранения данных в файл произвольного формата, с помощью отдельного класса, в формате, определяемом в соответствии с вариантом задания в пункте 5.1.

5) Для обработки файла создать отдельный класс/классы, реализующий методы загрузки из файла (возвращающий считанные данных в определенном формате) и сохранения в файл (сохраняющий матрицу в определенном формате, который может отличаться от внутреннего представления графа).

6) Протестировать класс для работы с заданным форматом файлов (при тестировании класса использовать подмену класса на StringReader /StringWriter для доступа к строковому потоку с последующей проверкой поученной выходной строки).

7) Протестировать класс, реализующий непосредственные вычисления (при получении данных и сохранения использовать заглушки и фиктивные объекты).

8) Оформить отчёт.

**Ход работы**

1-2) Создал класс, реализующий граф, хранящий информацию о структуре графа в формате матрицы инцидентности, а также реализован метод проверки графа на связность.

class Graph {

    constructor(incidenceMatrix) {

        this.incidenceMatrix = incidenceMatrix;

        this.numVertices = incidenceMatrix.length;

        this.numEdges = incidenceMatrix[0] ? incidenceMatrix[0].length : 0;

    }

    toEdgeList() {

        const edgeList = [];

        for (let e = 0; e < this.numEdges; e++) {

            let v1 = -1, v2 = -1;

            for (let v = 0; v < this.numVertices; v++) {

                if (this.incidenceMatrix[v][e] === 1) {

                    if (v1 === -1) v1 = v;

                    else if (v2 === -1) v2 = v;

                }

            }

            if (v1 !== -1 && v2 !== -1) {

                edgeList.push([v1, v2]);

            }

        }

        return edgeList;

    }

    isConnected() {

        if (this.numVertices === 0) return true;

        const visited = new Array(this.numVertices).fill(false);

        const adjList = this.toEdgeList().reduce((acc, [v1, v2]) => {

            acc[v1].push(v2);

            acc[v2].push(v1);

            return acc;

        }, Array.from({ length: this.numVertices }, () => []));

        const dfs = (v) => {

            visited[v] = true;

            for (const neighbor of adjList[v]) {

                if (!visited[neighbor]) {

                    dfs(neighbor);

                }

            }

        };

        dfs(0);

        return visited.every(v => v === true);

    }

}

module.exports = Graph;

3-5) Реализовали класс, читающий матрицу смежности из файла (GraphReader - загрузка) и класс с возможностью сохранить данные в файл в формате списка связности (GraphWriter - сохранение). Они поддерживают как и .txt, так и .html форматы. Вход – матрица смежности, внутри – матрица инцидентности, выход – список связности. Проверили работоспособность данных классов.

const fs = require('fs');

const cheerio = require('cheerio');

class GraphReader {

    read(filePath) {

        const data = fs.readFileSync(filePath, 'utf8');

        const ext = filePath.split('.').pop();

        if (ext === 'txt') {

            return this.parseTxt(data);

        } else if (ext === 'html') {

            return this.parseHtml(data);

        } else {

            throw new Error('Unsupported file format');

        }

    }

    parseTxt(data) {

        const lines = data.trim().split('\n');

        return lines.map(line => line.split(' ').map(Number));

    }

    parseHtml(data) {

        const $ = cheerio.load(data);

        const table = $('table');

        if (!table.length) {

            throw new Error('Не найдена таблица в HTML документе');

        }

        const adjacencyMatrix = [];

        table.find('tr').each((i, row) => {

            const rowValues = [];

            $(row).find('td').each((j, cell) => {

                rowValues.push(parseInt($(cell).text(), 10));

            });

            if (rowValues.length) {

                adjacencyMatrix.push(rowValues);

            }

        });

        return adjacencyMatrix;

    }

}

module.exports = GraphReader;

const fs = require('fs');

class GraphWriter {

    write(filePath, edgeList) {

        const ext = filePath.split('.').pop();

        if (ext === 'txt') {

            this.writeTxt(filePath, edgeList);

        } else if (ext === 'html') {

            this.writeHtml(filePath, edgeList);

        } else {

            throw new Error('Unsupported file format');

        }

    }

    writeTxt(filePath, edgeList) {

        let output = '';

        for (const [v1, v2] of edgeList) {

            output += `${v1} ${v2}\n`;

        }

        fs.writeFileSync(filePath, output);

    }

    writeHtml(filePath, edgeList) {

        let html = '<table border="1">\n';

        for (const [v1, v2] of edgeList) {

            html += `  <tr><td>${v1}</td><td>${v2}</td></tr>\n`;

        }

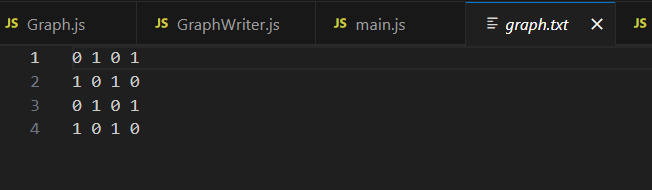
        html += '</table>';

        fs.writeFileSync(filePath, html);

    }

}

module.exports = GraphWriter;



const GraphReader = require('./GraphReader');

const GraphWriter = require('./GraphWriter');

const Graph = require('./Graph');

const reader = new GraphReader();

const adjacencyMatrix = reader.read('graph.txt');

const incidenceMatrix = Graph.adjacencyToIncidence(adjacencyMatrix);

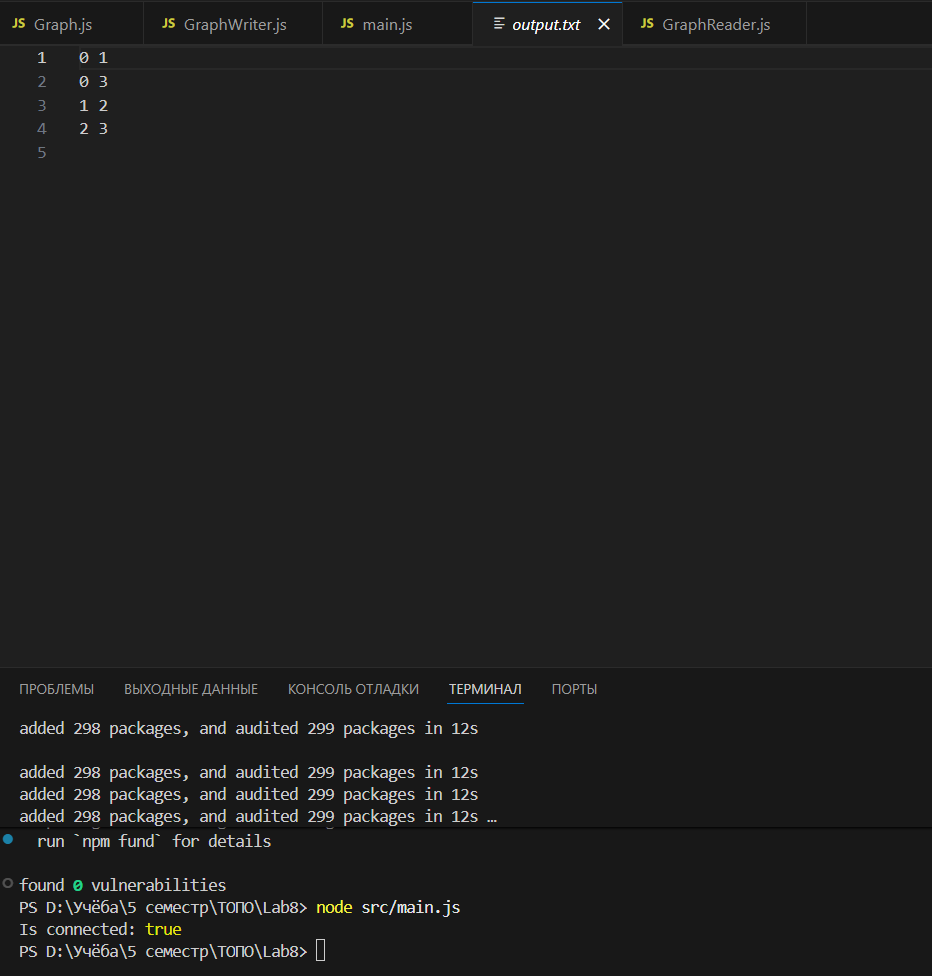
const graph = new Graph(incidenceMatrix);

console.log('Is connected:', graph.isConnected());

const edgeList = graph.toEdgeList();

const writer = new GraphWriter();

writer.write('output.txt', edgeList);



const GraphReader = require('./GraphReader');

const GraphWriter = require('./GraphWriter');

const Graph = require('./Graph');

const reader = new GraphReader();

const adjacencyMatrix = reader.read('graph.html');

const incidenceMatrix = Graph.adjacencyToIncidence(adjacencyMatrix);

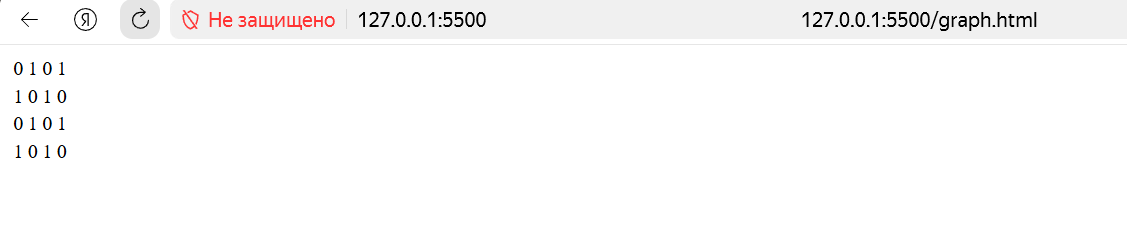
const graph = new Graph(incidenceMatrix);

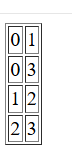
console.log('Is connected:', graph.isConnected());

const edgeList = graph.toEdgeList();

const writer = new GraphWriter();

writer.write('output.html', edgeList);





6-8) Провёл необходимые тесты с фиктивными объектами, с подменой на строковые потоки и составил отчёт.

const GraphReader = require('../src/GraphReader');

const mockReadTxt = (data) => {

    const reader = new GraphReader();

    const lines = data.trim().split('\n');

    return lines.map(line => line.split(' ').map(Number));

};

const mockReadHtml = (data) => {

    const reader = new GraphReader();

    const cheerio = require('cheerio');

    const $ = cheerio.load(data);

    const adjacencyMatrix = [];

    $('table tr').each((i, row) => {

        const rowValues = [];

        $(row).find('td').each((j, cell) => {

            rowValues.push(parseInt($(cell).text(), 10));

        });

        if (rowValues.length) {

            adjacencyMatrix.push(rowValues);

        }

    });

    return adjacencyMatrix;

};

test('GraphReader .txt', () => {

    const data = "0 1 0\n1 0 1\n0 1 0";

    const result = mockReadTxt(data);

    expect(result).toEqual([

        [0, 1, 0],

        [1, 0, 1],

        [0, 1, 0]

    ]);

});

test('GraphReader .html', () => {

    const data = `

        <table>

            <tr><td>0</td><td>1</td></tr>

            <tr><td>1</td><td>0</td></tr>

        </table>

    `;

    const result = mockReadHtml(data);

    expect(result).toEqual([

        [0, 1],

        [1, 0]

    ]);

});

const GraphWriter = require('../src/GraphWriter');

const mockWriteTxt = (edgeList) => {

    let output = '';

    for (const [v1, v2] of edgeList) {

        output += `${v1} ${v2}\n`;

    }

    return output;

};

const mockWriteHtml = (edgeList) => {

    let html = '<table border="1">\n';

    for (const [v1, v2] of edgeList) {

        html += `  <tr><td>${v1}</td><td>${v2}</td></tr>\n`;

    }

    html += '</table>';

    return html;

};

test('GraphWriter writes edge list to txt string', () => {

    const edgeList = [[0, 1], [1, 2]];

    const result = mockWriteTxt(edgeList);

    expect(result).toBe("0 1\n1 2\n");

});

test('GraphWriter writes edge list to html string', () => {

    const edgeList = [[0, 1], [1, 2]];

    const result = mockWriteHtml(edgeList);

    expect(result).toBe('<table border="1">\n  <tr><td>0</td><td>1</td></tr>\n  <tr><td>1</td><td>2</td></tr>\n</table>');

});

const Graph = require('../src/Graph');

const mockIncidenceMatrix = [

    [1, 0, 1],

    [1, 1, 0],

    [0, 1, 1]

];

test('Graph связный', () => {

    const graph = new Graph(mockIncidenceMatrix);

    expect(graph.isConnected()).toBe(true);

});

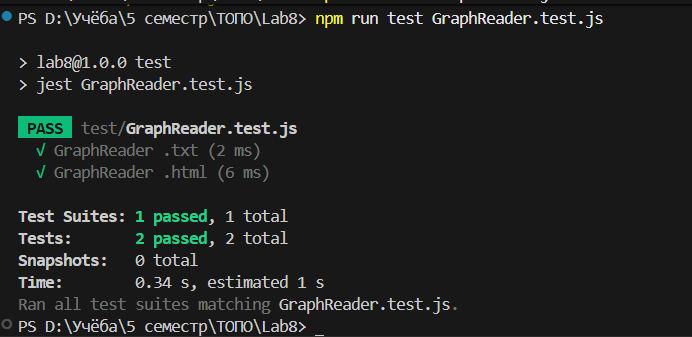
test('Graph корректный выходной формат - список связности', () => {

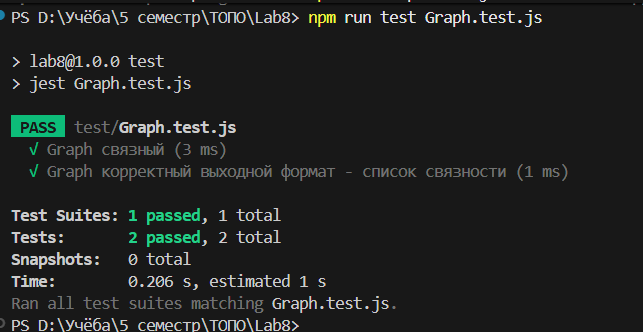
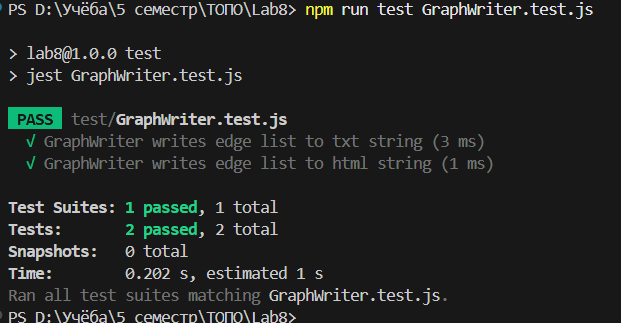
    const graph = new Graph(mockIncidenceMatrix);

    const edgeList = graph.toEdgeList();

    expect(edgeList).toEqual([[0, 1], [1, 2], [0, 2]]);

});





**Вывод**

В ходе выполнения данной лабораторной работы удалось изучить подход к автоматизации процесса тестирования с использованием тестовых двойников (Test Doubles).