Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Омский государственный технический университет»
Кафедра «Информатика и вычислительная техника»

ОТЧЁТ ПО РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЕ

по дисциплине «Проектирование и тестирование программного обеспечения»

«Пример оптимизации»

студента Овечкиной Екатерины Олеговны группы ПИН-231

Пояснительная записка

Шифр работы От-2068998-43-ПИН-37-23 ПЗ

Направление 09.03.04 Программная инженерия

Выполнил
Студент гр. ПИН-231
Овечкина Е.О. _____
(подп., дата)

Проверил
Старший преподаватель каф. ИВТ
Блохин А.В. _____
(подп., дата)

Омск 2025

Содержание

Введение

Оптимизация сетей представляет собой важную задачу, которая находит множество применений в самых различных областях, от социальных сетей до сложных инженерных систем. Под сетью обычно понимают структуру, состоящую из взаимосвязанных элементов, которые могут быть представлены в виде узлов и связей между ними.

В данной работе рассматривается процесс визуализации сетей с использованием методов оптимизации для анализа структуры и связей между узлами. Этот процесс может быть полезен не только в контексте социальных сетей, но и в других областях, таких как исследование коммуникационных потоков в организациях, анализ биологических сетей, а также в технических системах, где важно понимать взаимосвязь элементов и их взаимодействие. Особое внимание уделяется методам, которые позволяют выделить ключевые узлы и оптимизировать структуру сети для достижения максимальной эффективности.

1 Задача

В рамках работы «Пример оптимизации» необходимо разработать программу для визуализации сети с применением методов оптимизации для поиска центральных узлов и анализа их взаимосвязей, а также для улучшения структуры сети и выявления ключевых элементов.

Основные элементы задачи:

1. Моделирование сети: сеть состоит из узлов, связанных ребрами, которые могут представлять различные объекты, такие как люди в социальных сетях или устройства в технических системах. Связи показывают их взаимодействия и взаимозависимости.

2. Оптимизация структуры сети: применяются методы оптимизации для поиска ключевых узлов, играющих важную роль в сети, и анализа их значимости для функционирования всей сети.

3. Визуализация и анализ: визуализация сети позволяет наглядно представить её структуру, выявить важные узлы и анализировать связи. Программа поддерживает динамическую визуализацию, позволяя пользователю изменять структуру сети и следить за изменениями в реальном времени.

Задача нацелена на использование оптимизации для демонстрации различных методов анализа структуры сетей, а также на разработку инструментов для эффективного представления этих данных в визуальной форме.

Особенности задачи заключаются в том, программа должна обеспечивать динамическую визуализацию сети с возможностью добавления и удаления узлов и связей в реальном времени. Важным элементом является использование методов оптимизации для поиска центральных узлов. Программа должна сохранять данные сети и поддерживать анализ её структуры для улучшения связности. Интерфейс должен быть интерактивным, а архитектура — модульной для будущего расширения.

2 Особенности решения задачи

В этом разделе будет последовательно описано, как реализованы функции визуализации сети, алгоритмы, использованные в программе, а также как организованы взаимодействия с пользователем через интерфейс.

1. HTML (index.html)

В этой части кода создаётся структура веб-страницы. Основные элементы, которые необходимы для работы интерфейса:

– заголовок страницы: <h1> — используется для отображения на экране названия программы "Пример Визуализации Сети";

– контейнер для графа: — это место, где будет отображаться сам граф, визуализирующий сеть;

– кнопки управления: кнопки для добавления узлов, добавления связей между узлами, удаления узлов и связей, и кнопка для оптимизации сети, которая будет выделять центральные узлы.

На рисунке 1 представлен код, отвечающий за создание страницы. Этот код формирует структуру страницы перед взаимодействием с пользователем.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Визуализация Сети</title>
    <link rel="stylesheet" href="style.css">
    <script type="text/javascript" src="https://unpkg.com/vis-network@9.0.0/dist/vis-network.m
</head>
<body>
    <h1>Пример Визуализации Сети</h1>
    <div id="network"></div>

    <div class="controls">
        <button onclick="addNode()">Добавить узел</button>
        <button onclick="startAddingEdge()">Добавить связь</button>
        <button onclick="removeNode()">Удалить узел</button>
        <button onclick="removeEdge()">Удалить связь</button>
        <button onclick="optimizeNetwork()">Оптимизировать сеть</button>
    </div>

    <script src="script.js"></script>

</body>
</html>
```

Рисунок 1 – Код, создающий структуру страницы до добавления узлов и связей.

2. CSS (style.css)

CSS стили описывают внешний вид страницы, а также форматируют элементы управления:

– основной стиль страницы: фон, шрифт и цвет текста;

– контейнер для графа: задаются размеры для графа, чтобы он правильно отображался на странице;

– кнопки управления: стили кнопок с эффектами наведения для улучшения взаимодействия с пользователем.

На рисунке 2 представлен код стилей CSS, который отвечает за оформление страницы. Этот код задаёт внешний вид элементов, таких как фон, шрифты, кнопки и контейнер для графа.

```css
body {
    font-family: Arial, sans-serif;
    background-color: #e6e6fa;
    color: #333;
}

h1 {
    text-align: center;
    font-size: 32px;
    color: #6a5acd;
}

#network {
    width: 80%;
    height: 500px;
    margin: 20px auto;
    border: 2px solid #ddd;
    background-color: #fff;
    border-radius: 8px;
}

.controls {
    display: flex;
    justify-content: center;
    gap: 10px;
    margin-top: 20px;
}

button {
    padding: 10px 15px;
    background-color: #6a5acd;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #5a4eae;
}

```

Рисунок 2 – Код CSS, отвечающий за оформление элементов управления и графа на странице.

3. JavaScript (script.js)

JavaScript содержит основной функционал программы, включая логику работы с графом, добавление/удаление узлов и связей, а также оптимизацию сети.

– Инициализация данных.

В начале создаются узлы и связи для визуализации сети. С помощью библиотеки vis.js создаём граф, который можно отображать на веб-странице. На рисунке 3 представлен код инициализации данных для графа, включая создание узлов и связей с использованием библиотеки vis.js.

```
//инициализация данных для графа
let nodes = new vis.DataSet([
    { id: 1, label: 'Узел 1' },
    { id: 2, label: 'Узел 2' },
    { id: 3, label: 'Узел 3' },
]);

let edges = new vis.DataSet([
    { from: 1, to: 2 },
    { from: 2, to: 3 },
]);

//инициализация графа
let container = document.getElementById('network');
let data = {
    nodes: nodes,
    edges: edges
};
let options = {
    autoResize: true,
    nodes: {
        shape: 'dot',
        size: 10
    },
    physics: {
        enabled: true
    },
    interaction: {
        hover: true
    }
};
let network = new vis.Network(container, data, options);
console.log("Network initialized");
```

Рисунок 3 – Код JavaScript, отвечающий за инициализацию данных для графа, создание узлов и связей с использованием библиотеки vis.js.

– Функции добавления узлов и связей.

Функции addNode и startAddingEdge позволяют добавлять новые узлы и связи между ними. При этом каждому новому узлу присваивается уникальный ID.

На рисунке 4 представлен код функций добавления узлов и связей, который позволяет добавлять новые узлы и связи, присваивая каждому узлу уникальный ID.

```
39  //функция для добавления узла
40  function addNode() {
41      //получаем максимальный текущий ID
42      let maxNodeId = Math.max(...nodes.get().map(node => node.id), 0);
43
44      //создаем новый ID, увеличив максимальный
45      let newNodeId = maxNodeId + 1;
46
47      //добавляем новый узел с новым ID
48      nodes.add({ id: newNodeId, label: `Узел ${newNodeId}` });
49      console.log(`Добавлен Узел ${newNodeId}`);
50  }
51
52  //функция для начала добавления связи
53  function startAddingEdge() {
54      isAddingEdge = true;
55      alert('Выберите два узла для создания связи. Сначала кликните на один узел.');
56  }
```

Рисунок 4 – Код функций добавления узлов и связей, реализующий добавление новых элементов в граф и присваивающий каждому узлу уникальный ID.

– Функция оптимизации сети.

Функция optimizeNetwork находит центральный узел сети и выделяет его красным цветом, чтобы продемонстрировать алгоритм оптимизации. Связи остаются без изменений.

На рисунке 5 представлен код функции оптимизации сети, которая выделяет центральный узел красным цветом, не изменяя цвет связей.

```
111  //функция для оптимизации сети
112  function optimizeNetwork() {
113      //восстановим все узлы в исходный цвет
114      nodes.forEach(node => {
115          //проверяем, если узел уже красный, не меняем его
116          if (node.color && node.color.background === 'red') return;
117          nodes.update({ id: node.id, color: { background: '#97C2FC', border: '#2B7CE9' } });
118      });
119
120      let degrees = {};
121      edges.forEach(edge => {
122          degrees[edge.from] = (degrees[edge.from] || 0) + 1;
123          degrees[edge.to] = (degrees[edge.to] || 0) + 1;
124      });
125
126      let centralNode = Object.keys(degrees).reduce((a, b) => degrees[a] > degrees[b] ? a : b);
127      alert(`Центральный узел: Узел ${centralNode}`);
128
129      //изменим только цвет центрального узла, оставив его подпись и не трогая связи
130      nodes.update({ id: centralNode, color: { background: 'red', border: 'darkred' }, label: `
131
132      //связи не меняем (не изменяем их цвет)
133      edges.forEach(edge => {
134          edges.update({ id: edge.id, color: { color: '#2B7CE9' } });  // Связи остаются с исхо
135      });
136  }
```

Рисунок 5 – Код функции оптимизации сети, которая находит центральный узел и выделяет его красным цветом, оставляя связи без изменений.

В данном этапе отчёта были рассмотрены ключевые функции программы, реализующие визуализацию и оптимизацию сети. Мы подробно описали код, который отвечает за создание структуры страницы с помощью HTML и CSS, а также за функциональность на стороне JavaScript. В частности, были представлены функции, позволяющие добавлять узлы и связи, а также оптимизировать структуру сети путём выделения центрального узла. Также была продемонстрирована работа с библиотекой vis.js для визуализации данных и алгоритмов оптимизации. В результате был сформирован интерактивный инструмент для динамической работы с графом, который позволяет пользователю изменять структуру сети в реальном времени.

3 Тестирование

В ходе выполнения работы были проведены тесты программы, чтобы проверить корректность работы всех функций и алгоритмов. Для тестирования использовались различные сценарии, включая добавление узлов, создание связей, удаление элементов, а также оптимизация сети с выделением центрального узла.

На рисунке 6 показан экран программы после первого запуска, отображающий начальную структуру сети с несколькими узлами и связями.

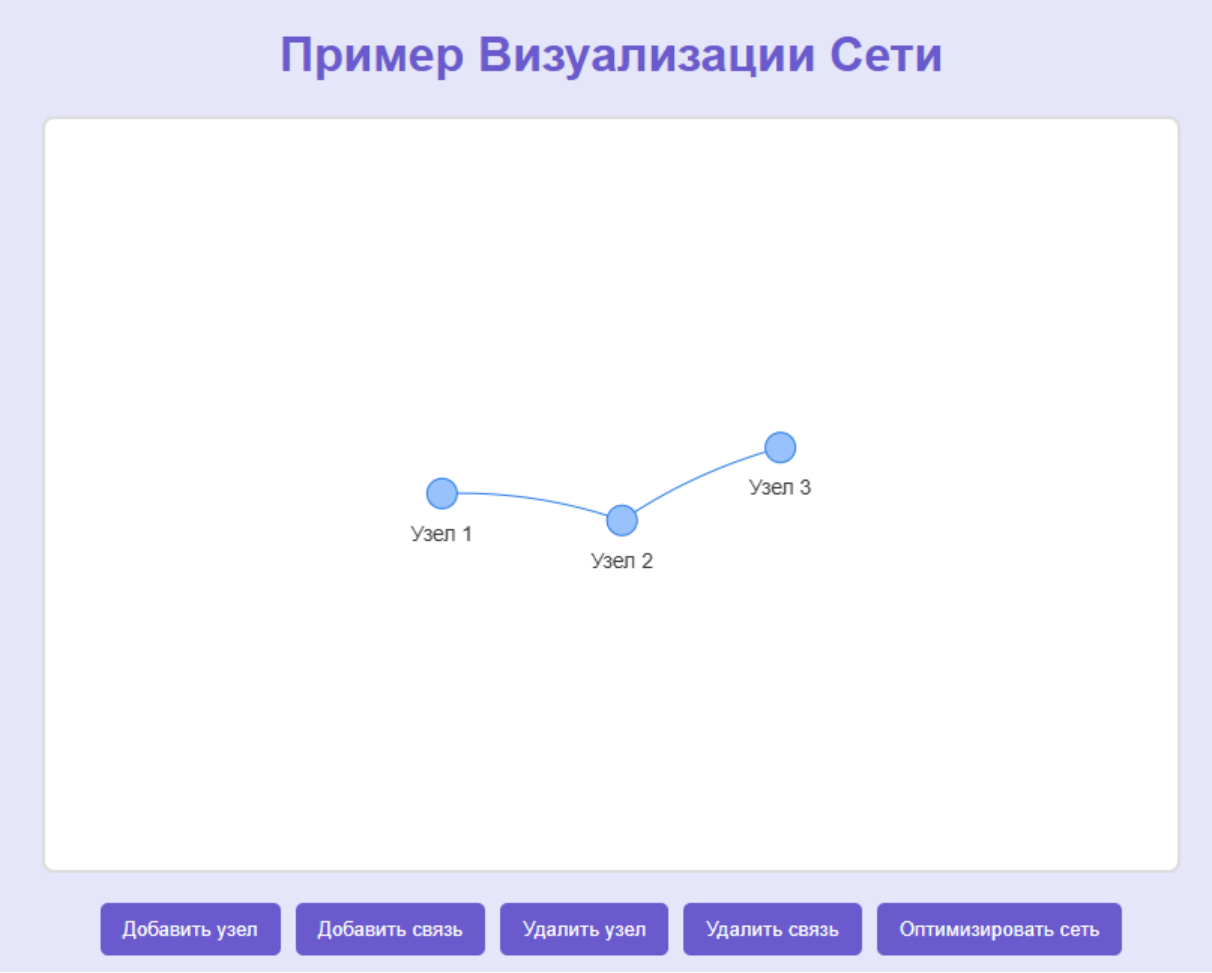

Рисунок 6 – Экран программы после первого запуска, отображающий начальную структуру сети с несколькими узлами и связями.

На данном этапе можно наблюдать, что программа корректно инициализирует граф с начальными данными. Все элементы управления активны, и пользователь может начать взаимодействовать с графом.

На рисунке 7 показан экран программы после добавления нескольких узлов и связей, демонстрируя процесс расширения сети.

Рисунок 7 – Экран программы после добавления нескольких узлов и связей, демонстрируя процесс расширения сети.

Здесь можно увидеть, как добавление новых узлов и связей работает без сбоев, и структура сети изменяется в реальном времени. Это подтверждает, что функции добавления узлов и связей работают корректно.

На рисунке 8 показан экран программы после выполнения функции оптимизации сети, с выделением центрального узла.

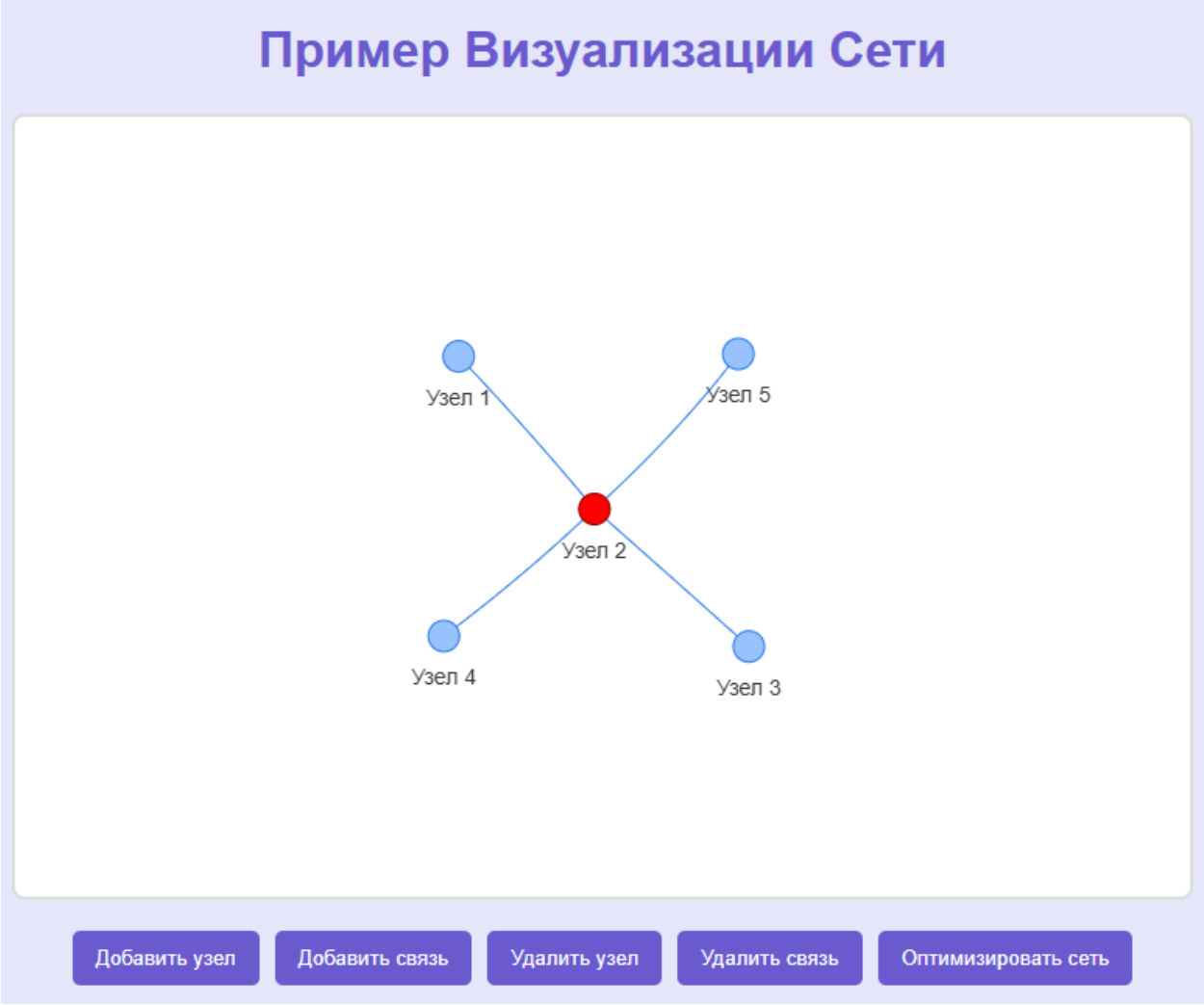

Рисунок 8 – Экран программы после выполнения функции оптимизации сети, с выделением центрального узла.

После выполнения функции оптимизации, программа находит центральный узел и выделяет его красным цветом. Связи остаются без изменений, что демонстрирует правильность работы алгоритма оптимизации.

На рисунке 9 показан экран программы при удалении одного узла, с автоматическим удалением всех связанных с ним связей.

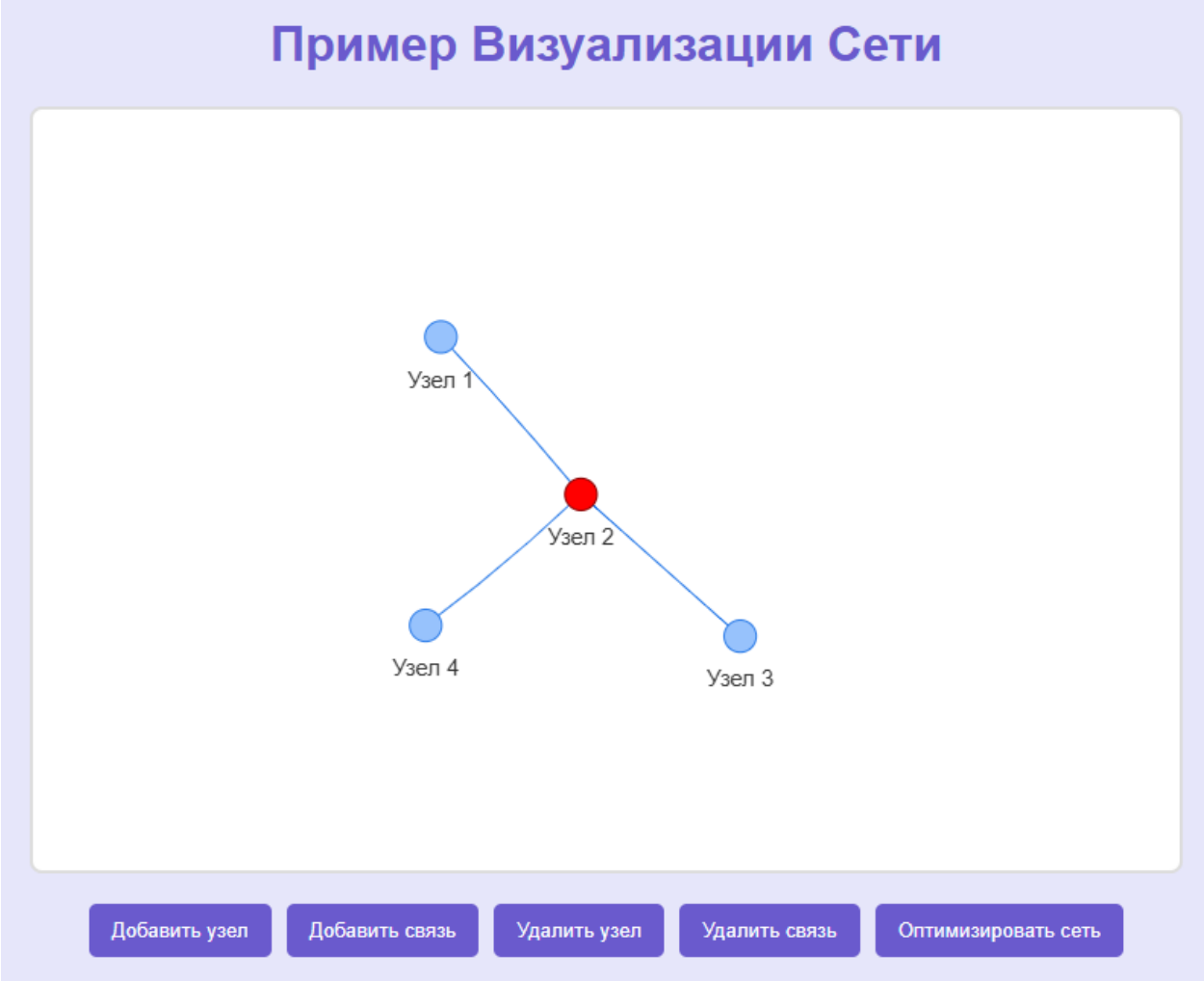

Рисунок 9 – Экран программы при удалении одного узла, с автоматическим удалением всех связанных с ним связей.

В этом тесте программа корректно удаляет узел и его связи, что подтверждает правильность работы функции удаления узлов и связей.

На рисунке 10 показан экран программы, где после добавления новых узлов и связей была выполнена повторная оптимизация сети, с сохранением всех изменений и корректной визуализацией.

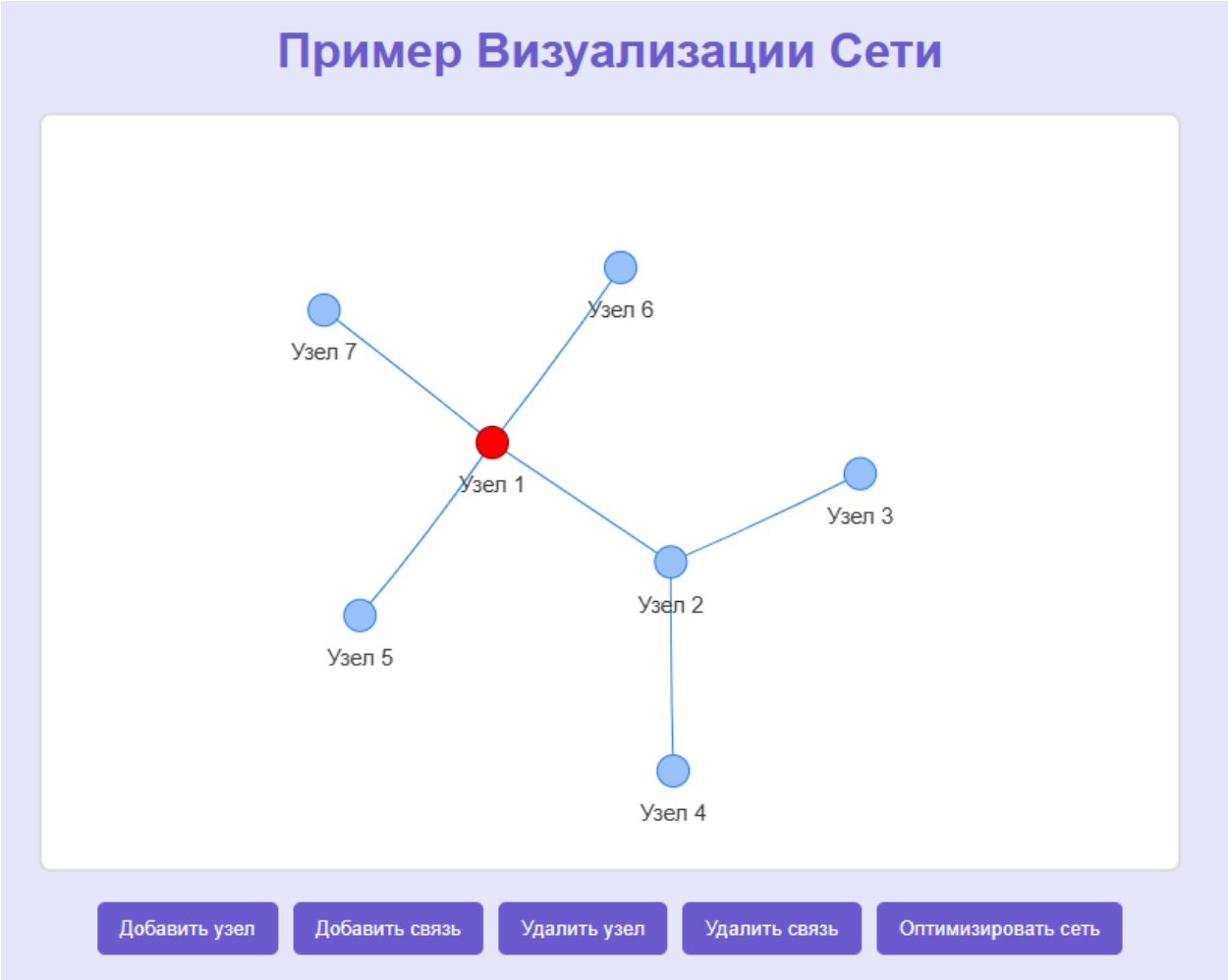

Рисунок 10 – Экран программы, где после добавления новых узлов и связей была выполнена повторная оптимизация сети, с сохранением всех изменений и корректной визуализацией.

В этом тесте программа корректно добавляет новые узлы и связи, а также выполняет повторную оптимизацию сети, что подтверждает правильность работы функций добавления элементов и алгоритма оптимизации.

В процессе тестирования были получены результаты, подтверждающие правильную работу программы в различных сценариях. Все функции – добавление, удаление и оптимизация узлов и связей – работают корректно, что подтверждается результатами тестов на различных данных.

Заключение

В ходе выполнения расчетно-графической работы по теме "Пример оптимизации" была разработана программа, которая реализует динамическую визуализацию и оптимизацию сетевой структуры. С использованием языка JavaScript и библиотеки vis.js была создана интерактивная сеть, включающая узлы и связи между ними. Программа позволяет добавлять и удалять узлы и связи, а также выполнять оптимизацию для выделения центральных узлов.

В результате работы была получена функциональная программа, демонстрирующая процесс изменения структуры сети в реальном времени и позволяющая оптимизировать её для поиска ключевых узлов. Эта программа может быть полезной для анализа различных типов сетевых структур, оптимизации связей и анализа взаимосвязей между элементами сети.

Таким образом, разработанная программа в рамках работы "Пример оптимизации" открывает возможности для изучения и оптимизации сетевых систем, предоставляя удобный инструмент для анализа структуры и динамики сетевых взаимодействий.

Список использованной литературы

1. Программируем коллективный разум. Доступно по адресу: https://cstor.nn2.ru/forum/data/forum/images/2012-12/59876050-programmiruem_kollektivniy_razum.pdf.

2. Петров, С. В. Алгоритмы и структуры данных: учебник / С. В. Петров. — М.: Наука, 2016. — 352 с.

3. Веб-ресурс *Vis.js Documentation*. Документация для визуализации графов. Доступно по адресу: https://visjs.org/docs/network/.

4. GitHub. Репозиторий библиотеки vis-network. Доступно по адресу: https://github.com/visjs/vis-network.

5. Абрамов, П. А. Программирование на JavaScript: учебник. — М.: Диалог-МГ, 2015. — 284 с.

6. Харитонов, А. И. Визуализация и анализ сетевых структур: методология и практика / А. И. Харитонов. — СПб.: БХВ-Петербург, 2018. — 400 с.

7. Рыбкин, В. В. Алгоритмы поиска оптимальных путей в сетях / В. В. Рыбкин. — М.: Издательство МГУ, 2017. — 310 с.

8. Шмидт, Т. Т. Теория графов и её применение в информатике / Т. Т. Шмидт. — СПб.: Питер, 2014. — 256 с.

9. Колесников, Д. А. Визуализация данных с использованием библиотеки vis.js. Доступно по адресу: https://habr.com/ru/post/313272/.

10. Ковалев, М. В. Программирование на языке JavaScript. — М.: Инфра-М, 2016. — 240 с.

11. Головин, Е. С. Модели и методы оптимизации сетевых структур / Е. С. Головин. — М.: Высшая школа, 2019. — 312 с.

12. Википедия. Теория графов. Доступно по адресу: https://ru.wikipedia.org/wiki/Теория_графов.

13. Веб-сайт International Society for Artificial Life. Доступно по адресу: http://www.alife.org.

14. Смирнов, А. П. Практическое руководство по разработке веб-

приложений с использованием JavaScript и HTML5. — М.: ДМК Пресс, 2017. — 264 с.

15. Фадеев, В. А. Алгоритмы и структуры данных. — М.: БХВ-Петербург, 2015. — 320 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Программный код (index.html)

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Визуализация Сети</title>
    <link rel="stylesheet" href="style.css">
    <script type="text/javascript" src="https://unpkg.com/vis-
network@9.0.0/dist/vis-network.min.js"></script>
</head>
<body>
    <h1>Пример Визуализации Сети</h1>
    <div id="network"></div>

    <div class="controls">
        <button onclick="addNode()">Добавить узел</button>
        <button onclick="startAddingEdge()">Добавить связь</button>
        <button onclick="removeNode()">Удалить узел</button>
        <button onclick="removeEdge()">Удалить связь</button>
        <button onclick="optimizeNetwork()">Оптимизировать сеть</button>
    </div>

    <script src="script.js"></script>

</body>
</html>
```

ПРИЛОЖЕНИЕ В
(обязательное)
Программный код (style.css)

```
body {
    font-family: Arial, sans-serif;
    background-color: #e6e6fa;
    color: #333;
}

h1 {
    text-align: center;
    font-size: 32px;
    color: #6a5acd;
}

#network {
    width: 80%;
    height: 500px;
    margin: 20px auto;
    border: 2px solid #ddd;
    background-color: #fff;
    border-radius: 8px;
}

.controls {
    display: flex;
    justify-content: center;
    gap: 10px;
    margin-top: 20px;
}

button {
    padding: 10px 15px;
    background-color: #6a5acd;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
button:hover {
    background-color: #5a4eae;
}
```

ПРИЛОЖЕНИЕ С
(обязательное)
Программный код (script.js)

```
//инициализация данных для графа
let nodes = new vis.DataSet([
    { id: 1, label: 'Узел 1' },
    { id: 2, label: 'Узел 2' },
    { id: 3, label: 'Узел 3' },
]);

let edges = new vis.DataSet([
    { from: 1, to: 2 },
    { from: 2, to: 3 },
]);

//инициализация графа
let container = document.getElementById('network');
let data = {
    nodes: nodes,
    edges: edges
};
let options = {
    autoResize: true,
    nodes: {
        shape: 'dot',
        size: 10
    },
    physics: {
        enabled: true
    },
    interaction: {
        hover: true
    }
};
let network = new vis.Network(container, data, options);
console.log("Network initialized");

let selectedNodeId = null;
let isAddingEdge = false;
let edgeStartNodeId = null;

//функция для добавления узла
```

```
function addNode() {
    //получаем максимальный текущий ID
    let maxNodeId = Math.max(...nodes.get().map(node => node.id), 0);

    //создаем новый ID, увеличив максимальный
    let newNodeId = maxNodeId + 1;

    //добавляем новый узел с новым ID
    nodes.add({ id: newNodeId, label: `Узел ${newNodeId}` });
    console.log(`Добавлен Узел ${newNodeId}`);
}

//функция для начала добавления связи
function startAddingEdge() {
    isAddingEdge = true;
    alert('Выберите два узла для создания связи. Сначала кликните на один
узел.');
}

//функция для добавления связи между двумя узлами
network.on('selectNode', function(event) {
    if (isAddingEdge) {
        if (edgeStartNodeId === null) {
            //выбираем первый узел
            edgeStartNodeId = event.nodes[0];
            nodes.update({ id: edgeStartNodeId, color: { border: 'red' } });
//подсветка выбранного узла
            console.log(`Выбран первый узел: Узел ${edgeStartNodeId}`);
        } else {
            //выбираем второй узел и добавляем связь
            let edgeEndNodeId = event.nodes[0];
            if (edgeStartNodeId !== edgeEndNodeId) {
                edges.add({ from: edgeStartNodeId, to: edgeEndNodeId });
                console.log(`Добавлена связь между Узлом ${edgeStartNodeId} и
Узлом ${edgeEndNodeId}`);
            }
            //снимаем подсветку
            nodes.update({ id: edgeStartNodeId, color: { border: '#6a5acd' }
});
            edgeStartNodeId = null;
            isAddingEdge = false;
        }
    }
```

```
});

//функция для удаления узла
function removeNode() {
    if (selectedNodeId === null) {
        alert('Выберите узел, который хотите удалить!');
        return;
    }

    nodes.remove({ id: selectedNodeId });
    edges.get().forEach(edge => {
        if (edge.from === selectedNodeId || edge.to === selectedNodeId) {
            edges.remove(edge);
        }
    });

    console.log(`Удален Узел ${selectedNodeId}`);
    selectedNodeId = null;  //сбрасываем выбранный узел
}

//функция для удаления связи
function removeEdge() {
    let selectedEdges = network.getSelectedEdges();
    if (selectedEdges.length === 0) {
        alert('Выберите связь для удаления!');
        return;
    }

    edges.remove({ id: selectedEdges[0] });
    console.log(`Удалена связь: ${selectedEdges[0]}`);
}

//функция для оптимизации сети
function optimizeNetwork() {
    //восстановим все узлы в исходный цвет
    nodes.forEach(node => {
        //проверяем, если узел уже красный, не меняем его
        if (node.color && node.color.background === 'red') return;
        nodes.update({ id: node.id, color: { background: '#97C2FC', border:
'#2B7CE9' } });
    });

    let degrees = {};
```

```
    edges.forEach(edge => {
        degrees[edge.from] = (degrees[edge.from] || 0) + 1;
        degrees[edge.to] = (degrees[edge.to] || 0) + 1;
    });

    let centralNode = Object.keys(degrees).reduce((a, b) => degrees[a] > degrees[b] ? a : b);
    alert(`Центральный узел: Узел ${centralNode}`);

    //изменим только цвет центрального узла, оставив его подпись и не трогая связи
    nodes.update({ id: centralNode, color: { background: 'red', border: 'darkred' }, label: `Узел ${centralNode}` });

    //связи не меняем (не изменяем их цвет)
    edges.forEach(edge => {
        edges.update({ id: edge.id, color: { color: '#2B7CE9' } });  // Связи остаются с исходным цветом
    });
}


//обработчик клика по узлу для выбора
network.on('selectNode', function(event) {
    selectedNodeId = event.nodes[0];
    console.log(`Выбран Узел ${selectedNodeId}`);
});
```