**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра** **ВТ**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Списки»**

**Вариант 5**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0322 |  | Кренев И.Ф. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2022

**Постановка задачи**

**Цель работы**

Получение практических навыков решения задач с использованием принципа «Разделяй и властвуй» и навыков определения асимптотической сложности решения.

**Формулировка задания**

Решить задачу с использованием принципа «Разделяй и властвуй». Определить теоретическую асимптотическую сложность решения. Эмпирически оценить временную сложность решения для «average-case».

О влюбленных улитках. На квадрате земли 1 км2 располагаются улитки-гермафродиты. В момент времени каждая из улиток с постоянной скоростью 1 cм/с ползет к улитке, являющейся ближайшей к ней в момент времени t = 0, выбрав её в качестве спутника жизни. Определить время, через которое первая пара улиток достигнет друг друга или наличие ситуации, приводящей улиток в замешательство, наиболее эффективным способом**Описание программы**

Созданная программа реализует 4 основных функции (findClosestDistance, brute, dist, minPair), отвечающих за нахождение дистанции между точками и вычисление времени и 7 побочных (clear, circle, init, getOutput, changeText, generateRandomly), отвечающих за отрисовку данных.

Описание функций:

* findClosestDistance – основная функция программы, принимающая на вход массив координат точек и возвращающая пару точек, расстояние между которыми является наименьшим. Внутри функции используется рекурсия.
* brute – функция, использующаяся для поиска ближайших точек в случаях, когда основная функция для поиска не сможет корректно рассчитать результат.
* dist – функция, используемая для нахождения расстояния между точками;
* minPair – функция, используемая для определения наименьшего расстояния между двумя парами точек.
* cicle – функция, отвечающая за отрисовку поля с точками;
* clear – функция очистки поля;
* init – функция, отвечающая за инициализацию программы;
* getOutput – функция вывода результата выполнения программы;
* generateRandomly – функция, генерирующая случайный набор точек;
* changeText – функция, отвечающая за смену точек в поле для ввода.

**Теоретическая асимптотическая сложность**

Перед началом анализа асимптотической сложности алгоритма стоит заметить, что исходя из вводных данных задача может быть решена двумя способами – нерекурсивно (если количество точек меньше или равно трем) и рекурсивно (во всех остальных случаях). Под количеством точек будет принимать n.

В процессе рекурсивной обработки данных задача разбивается на несколько подзадач (а), размер каждой из которых равен n/b. Вызов каждого узла рекурсии в этом случае равен соотношению f(n). Общий объем работы алгоритма определяется как сумма всех работ в каждом узле.

Воспользуемся основной теоремой о рекуррентных соотношениях, которая рассматривает следующие отношения:

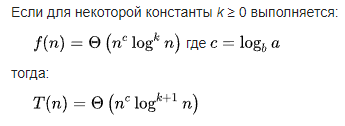


где:

* n — размер задачи.
* a — количество подзадач в рекурсии.
* n/b — размер каждой подзадачи. (Предполагается, что все подзадачи на каждом этапе имеют одинаковый размер.)
* f (n) — оценка сложности работы, производимой алгоритмом вне рекурсивных вызовов. В неё также включается вычислительная стоимость деления на подзадачи и объединения результатов решения подзадач.

Для начала необходимо определить сложность не рекурсивной части задачи. Для не рекурсивных расчетов используется только одна функция – brute. Из-за наличия в ней цикла, вложенного в другой цикл, сложность составляет O(n2), то есть f(n) = n2.

Попробуем воспользоваться общей формой теоремы:



Для примера будем считать, что у нас имеется 8 точек, расположенных по 4 с каждой стороны от центра плоскости. Поскольку функция разбивает плоскость на 2 части, мы будем иметь *a* и *b*, равное 2.

**Пример работы программы**

Интерфейс веб-страницы с программой содержит два основных раздела –связанный список и набор операций, взаимодействие с которыми осуществляется по нажатию соответствующих кнопок:

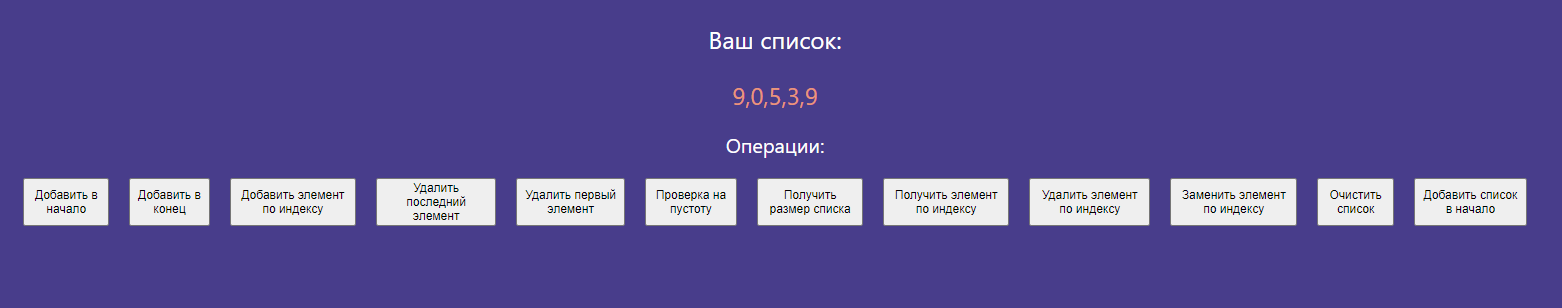


Рисунок – основное меню программы

При выполнении операций, требующих ввод данных пользователя, открывается модальное окно, содержащее сообщение и поля для ввода данных. Количество полей зависит от количества аргументов функции. Поля ввода ограничены используемым типом данных - возможен ввод только целых чисел.

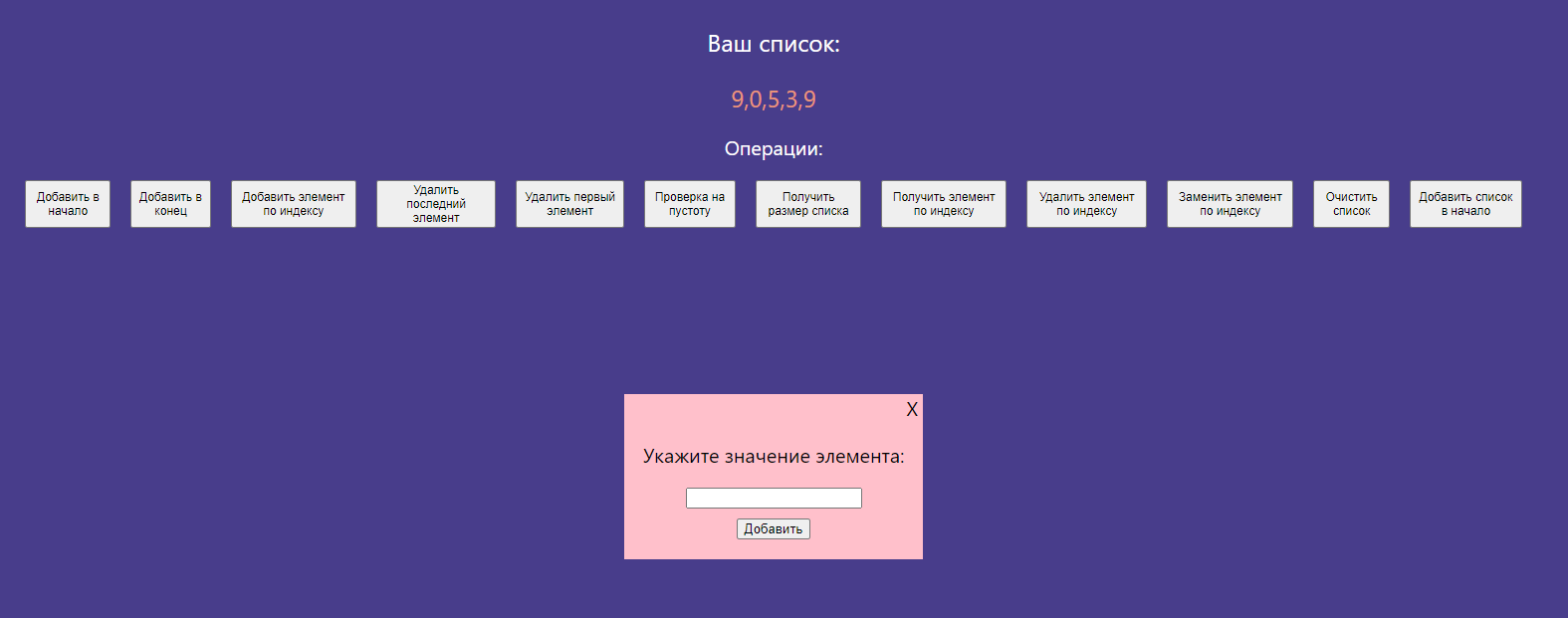


Рисунок 2 – модальное окно ввода одного параметра

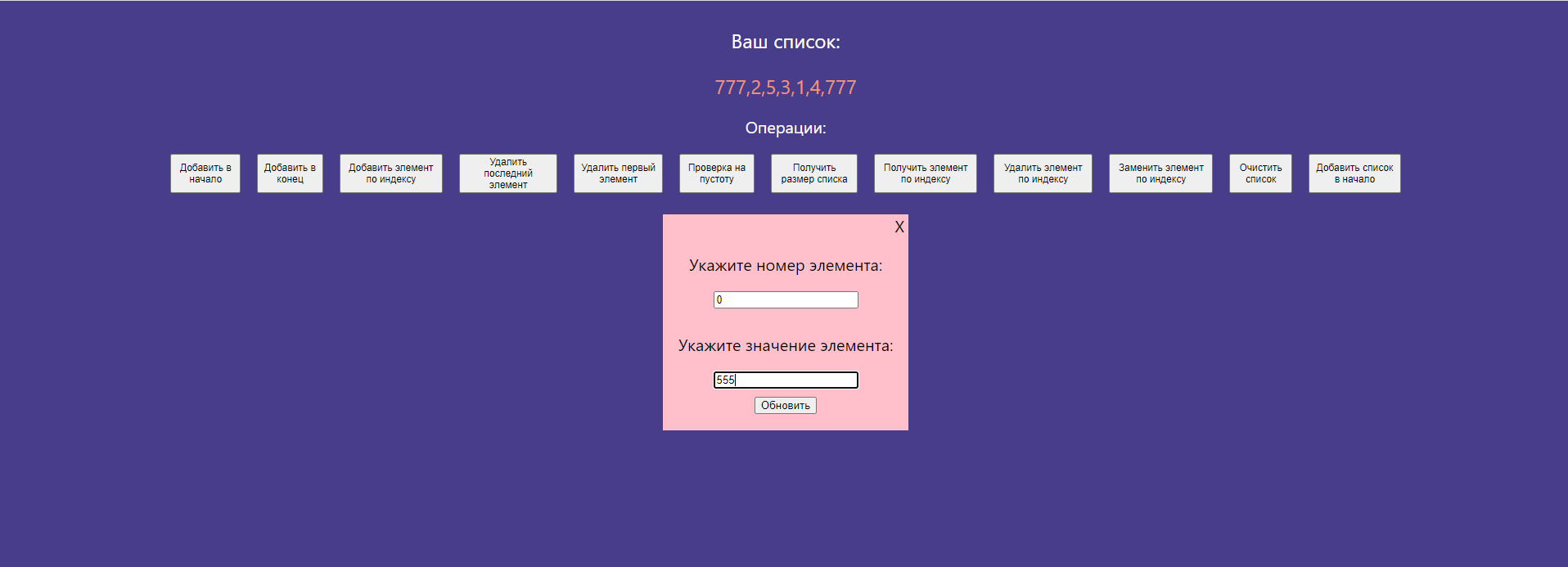


Рисунок 3 – модальное окно ввода двух параметров

В случае, если операция не требует ввода данных пользователя, результат операции будет отображен в модальном окне:

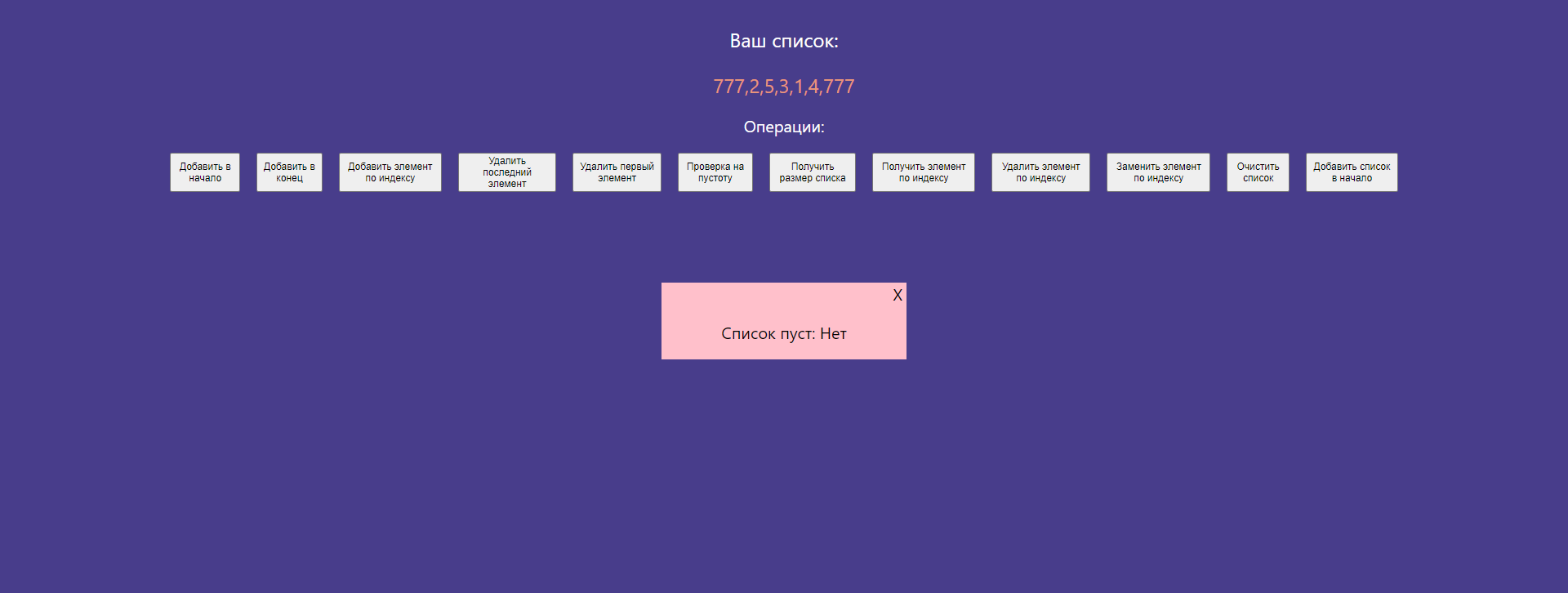


Рисунок 4 – модальное окно без параметров

После выполнении операции в нижнем правом углу экрана возникает уведомление о статусе её выполнения. Цвет уведомления зависит от статуса выполнения операции:

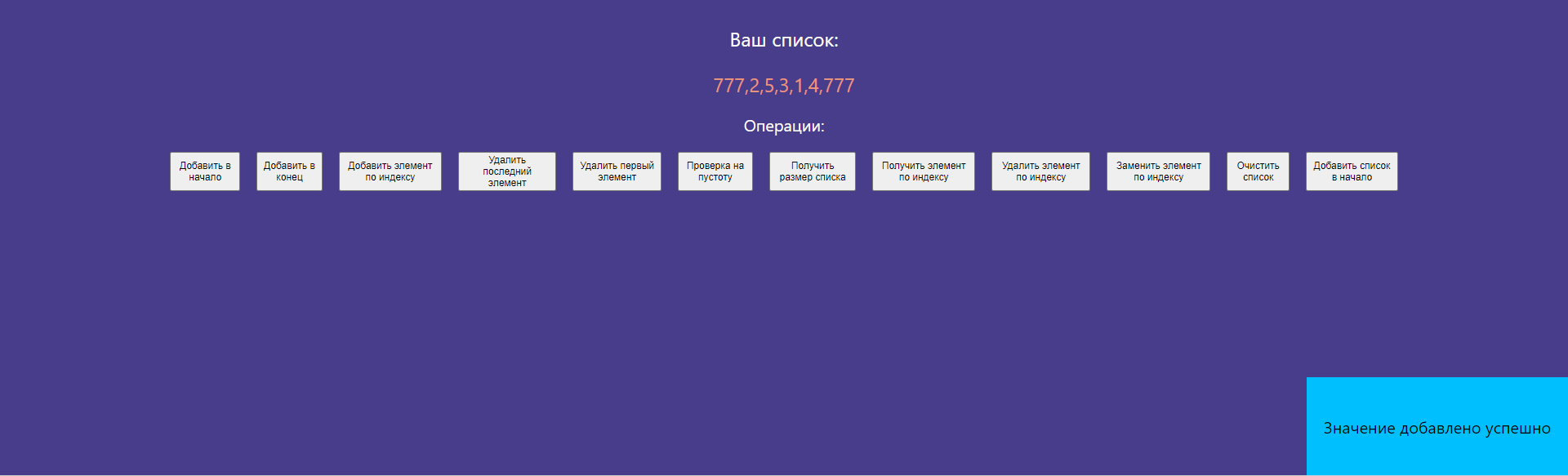


Рисунок 5 – уведомление об успешном выполнении операции

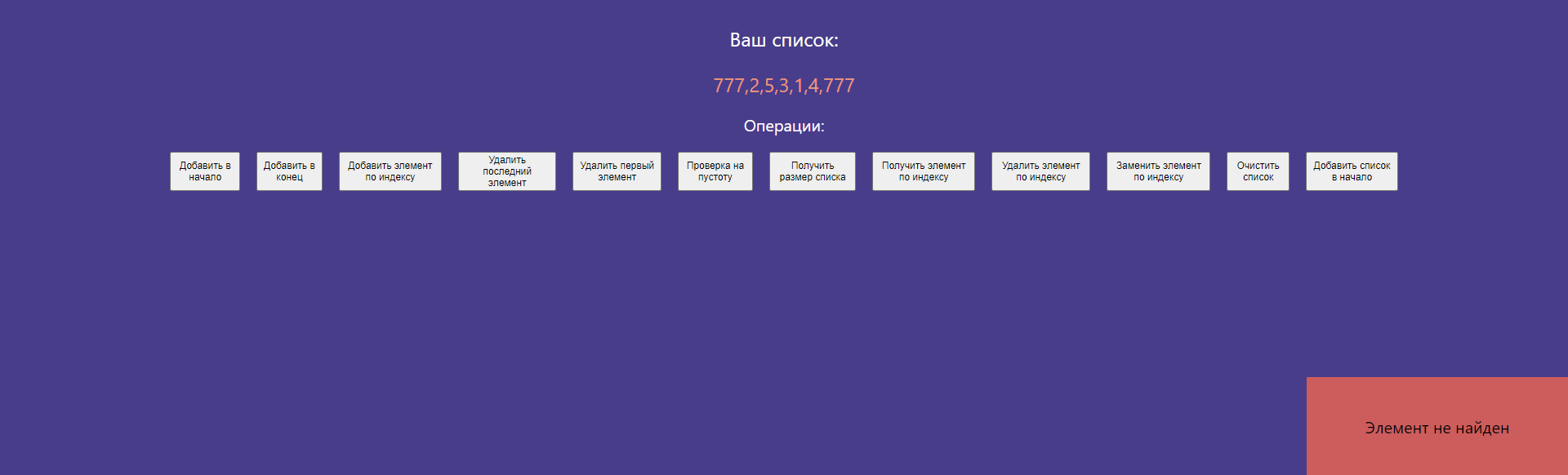


Рисунок 6 – уведомление о неуспешном выполнении операции

**Выводы**

В ходе выполнения лабораторной работы были приобретены навыки создания линейных списков и работы с ними. Были изучены методы добавления, удаления, замены, подсчета количества элементов связанного списка. Также были приобретены навыки по объединению нескольких списков в один.

**Приложение 1. Листинг программного кода**

Ссылка на GitHub: <https://github.com/IlyaKrenev/aisd1>

Ссылка на GitHub Pages: <https://ilyakrenev.github.io/aisd1/>

**list.js:**

class Dialog {

    constructor () {

        this.dialog = document.querySelector('.dialog');

    }

    show (*msg*, *isError* = false) {

        this.dialog.textContent = *msg*;

        this.dialog.style.display = 'flex';

        this.dialog.style.backgroundColor = *isError* ? 'indianred' : 'deepskyblue';

        setTimeout(() => {

            this.dialog.style.display = 'none'

        }, 3000)

    }

}

class ListNode {

    constructor(*value*, *next* = null) {

        this.value = *value*;

        this.next = *next*;

    }

}

class List {

    constructor (*callback*) {

        this.onChangeListFN = *callback*;

        this.\_head = null;

        this.dialog = new Dialog();

    }

    get head () {

*return* this.\_head;

    }

    set head (*value*) {

        this.\_head = *value*;

        this.onChangeListFN();

    }

    prepend (*value*) {

        const newNode = new ListNode(*value*, this.head);

        this.head = newNode;

        this.dialog.show('Значение добавлено успешно')

    }

    append (*value*, *needMessage* = true) {

        const newNode = new ListNode(*value*);

*if* (!this.head) {

            this.head = newNode;

*needMessage* && this.dialog.show('Значение добавлено успешно');

*return*;

        }

        let tempNode = this.head;

*while* (tempNode.next !== null) {

            tempNode = tempNode.next;

        }

        tempNode.next = newNode;

        this.onChangeListFN();

*if* (*needMessage*) {

            this.dialog.show('Значение добавлено успешно');

        }

    }

    deleteLast () {

*if* (this.head === null) {

            this.dialog.show('Список пуст', true);

*return*;

        }

*if* (this.head.next === null) {

            this.head = null;

            this.dialog.show('Значение удалено успешно');

*return*;

        }

        let prelastNode = this.head;

*while* (prelastNode.next !== null) {

*if* (prelastNode.next.next === null) {

                prelastNode.next = null;

            }

*else* {

                prelastNode = prelastNode.next;

            }

        }

        this.onChangeListFN();

        this.dialog.show('Значение удалено успешно');

    }

    deleteFirst () {

*if* (!this.head) {

            this.dialog.show('Элемент не найден', true);

*return*;

        }

*if* (this.head.next) {

            this.head = this.head.next;

        }

*else* {

            this.head = null;

        }

        this.dialog.show('Значение удалено успешно');

    }

    isNull () {

*return* this.head === null;

    }

    getList () {

        let outArray = [];

        let node = this.head;

*if* (node !== null) {

*while* (node !== null) {

                outArray.push(node.value);

                node = node.next;

            }

        }

*return* outArray;

    }

    getSize () {

        let node = this.head;

        let size = 0;

*while* (node !== null) {

            size += 1;

            node = node.next;

        }

*return* size;

    }

    getElemByIndex (*index*) {

        let node = this.head;

*if* (node === null) {

            this.dialog.show('Элемент не найден', true);

*return*;

        }

*for* (let i = 0; i < Number(*index*); i++) {

*if* (node.next !== null) {

                node = node.next;

            } *else* {

                this.dialog.show('Элемент не найден', true);

*return*;

            }

        }

*return* node.value;

    }

    clearList () {

*while* (this.head !== null) {

            const node = this.head.next;

            this.head = null;

            this.head = node;

        }

        this.dialog.show('Список очищен');

    }

    insert(*index*, *value*) {

*if* (Number(*index*) === 0) {

            this.prepend(*value*);

*return*;

        }

        const newNode = new ListNode(*value*);

        let node = this.head;

*for* (let i = 0; i < Number(*index*) - 1; i++) {

*if* (node.next !== null) {

                node = node.next;

            } *else* {

                this.dialog.show('Элемент не найден', true);

*return*;

            }

        }

        newNode.next = node.next;

        node.next = newNode;

        this.dialog.show('Элемент успешно добавлен');

        this.onChangeListFN()

    }

    removeElemByIndex (*index*) {

*if* (this.head === null) {

            this.dialog.show('Элемент не найден', true);

*return*;

        }

*if* (Number(*index*) === 0){

            this.deleteFirst();

*return*;

        }

        let node = this.head;

        let current = this.head;

*for* (let i = 1; i < Number(*index*); i++) {

*if* (node.next !== null) {

                current = node;

                node = node.next;

            } *else* {

                this.dialog.show('Элемент не найден', true);

*return*;

            }

        }

        node = current.next;

        current.next = current.next.next;

        node = null;

        this.dialog.show('Элемент успешно удален');

        this.onChangeListFN()

    }

    updateElemByIndex (*index*, *value*) {

        let node = this.head;

*if* (node === null) {

            this.dialog.show('Элемент не найден', true);

*return*;

        }

*for* (let i = 0; i < Number(*index*) - 1; i++) {

*if* (node.next !== null) {

                node = node.next;

            } *else* {

                this.dialog.show('Элемент не найден', true);

*return*;

            }

        }

        node.value = *value*;

        this.dialog.show('Элемент успешно обновлен');

        this.onChangeListFN()

    }

    reverseList (*list*) {

        let reversedList = null;

*for* (let node = *list*; node !== null; node = node.next) {

            node.next = reversedList;

            reversedList = node;

        }

*return* reversedList;

    }

    prependList () {

        let newList = new List(() => {});

*for* (let i = 0; i < 5; i++) {

            newList.append(Math.floor(Math.random() \* 10), false);

        }

        const newListValue = newList.getList();

*if* (this.head === null) {

            this.head = newList.head;

            this.dialog.show(`Новый список (${newListValue}) успешно добавлен в начало основного`);

*return*;

        }

        let current = newList.head;

        let prev = null;

        let next = null;

*while* (current !== null) {

            next = current.next;

            current.next = prev;

            prev = current;

            current = next;

        }

        newList.head = prev;

        let reversedListNode = newList.head;

*while* (reversedListNode !== null) {

            this.prepend(reversedListNode.value, false);

            reversedListNode = reversedListNode.next;

        }

        newList = null;

        this.dialog.show(`Новый список (${newListValue}) успешно добавлен в начало основного`);

    }

    methodsOptions = {

        'prepend': {name: 'Добавить в начало', modal: true, header: 'Укажите значение элемента:', action: 'Добавить'},

        'append':  {name: 'Добавить в конец', modal: true, header: 'Укажите значение элемента:', action: 'Добавить'},

        'insert': {name: 'Добавить элемент по индексу', modal: true, header: 'Укажите номер элемента:', header2: 'Укажите значение элемента:', action: 'Добавить'},

        'deleteLast': {name: 'Удалить последний элемент', modal: false},

        'deleteFirst': {name: 'Удалить первый элемент', modal: false},

        'isNull': {name: 'Проверка на пустоту', modal: true, header: 'Список пуст:'},

        'getSize': {name: 'Получить размер списка', modal: true, header: 'Размер списка:'},

        'getElemByIndex': {name: 'Получить элемент по индексу', modal: true, header: 'Укажите номер элемента:', action: 'Найти', needAnswer: true, answer: 'Элемент: '},

        'removeElemByIndex': {name: 'Удалить элемент по индексу', modal: true, header: 'Укажите номер элемента:', action: 'Удалить'}, *//*

        'updateElemByIndex': {name: 'Заменить элемент по индексу', modal: true, header: 'Укажите номер элемента:', header2: 'Укажите значение элемента:', action: 'Обновить'}, *//*

        'clearList': {name: 'Очистить список', modal: false},

        'prependList': {name: 'Добавить список в начало', modal: false}

    }

}

**main.js:**

class Main {

    constructor () {

        this.listSpan = document.querySelector('.listSpan');

        this.modalWindow = document.querySelector('.modalWindow');

        this.linkedList = new List(this.updateListValue);

        this.modalWindowComponents = {

            modalHeader1: null,

            modalHeader2: null,

            modalInput1: null,

            modalInput2: null,

            modalButton: null

        }

        this.initButtons();

        this.initWindowComponents();

    }

    updateListValue = () => {

*if* (this.listSpan && this.linkedList !== null) {

            this.listSpan.textContent = this.linkedList.getList();

        }

    }

    initButtons = () => {

        const buttonsContainer = document.querySelector('.buttonsContainer');

        const methodsDictionary = Object.entries(this.linkedList.methodsOptions);

        methodsDictionary.forEach(([*methodName*, *methodsOptions*]) => {

            const button = document.createElement('button');

            button.textContent = *methodsOptions*.name;

*if* (*methodsOptions*.modal === false) {

                button.addEventListener('click', () => this.handleCommonClick(*methodName*));

            } *else* {

                button.addEventListener('click', () => this.openModal(*methodName*, *methodsOptions*));

            }

            buttonsContainer.append(button);

        });

        const closeModalButton = document.querySelector('.modalWindow\_closeModal');

        closeModalButton.onclick = () => {

            this.hideElement(this.modalWindow);

        }

    }

    validateInput = (*event*) => {

        console.log(*event*.target.value)

*event*.target.value = *event*.target.value.replace(/[^\d]/g, "");

    }

    initWindowComponents = () => {

        this.modalWindowComponents.modalHeader1 = document.querySelector('.modalWindow\_msg1\_header');

        this.modalWindowComponents.modalHeader2 = document.querySelector('.modalWindow\_msg2\_header');

        this.modalWindowComponents.modalButton = document.querySelector('.modalWindow\_action');

        const modalInput1 = document.querySelector('.modalWindow\_msg1\_input');

        const modalInput2 = document.querySelector('.modalWindow\_msg2\_input');

        modalInput1.onpaste = (*e*) => {

*e*.preventDefault();

        }

        modalInput2.onpaste = (*e*) => {

*e*.preventDefault();

        }

        modalInput1.oninput = (*e*) => {

            const value = Number(*e*.data);

*if* (!Number.isInteger(value)) {

                debugger

*e*.target.value = e.target.value.slice(0, -1);

            }

        }

        modalInput2.oninput = (*e*) => {

            const value = Number(*e*.data);

*if* (!Number.isInteger(value)) {

*e*.target.value = *e*.target.value.slice(0, -1);

            }

        }

        this.modalWindowComponents.modalInput1 = modalInput1;

        this.modalWindowComponents.modalInput2 = modalInput2;

    }

    handleCommonClick = (*methodName*) => {

        this.linkedList[*methodName*]();

    }

    clearInputs = () => {

        this.modalWindowComponents.modalInput1.value = '';

        this.modalWindowComponents.modalInput2.value = '';

    }

    openModal = (*methodName*, *methodsOptions*) => {

        this.showElement(this.modalWindow, true);

        const {header: methodHeader1, header2: methodHeader2, action: methodAction} = *methodsOptions*;

        const {modalHeader1, modalHeader2, modalInput1, modalInput2, modalButton} = this.modalWindowComponents;

        this.clearInputs();

        modalHeader1.textContent = methodHeader1;

        modalButton.textContent = methodAction;

*if* (methodHeader2 !== undefined) {

            this.showElement(modalHeader2)

            this.showElement(modalInput2)

            modalHeader2.textContent = methodHeader2

        } *else* {

            this.hideElement(modalHeader2);

            this.hideElement(modalInput2);

        }

*if* (methodAction !== undefined) {

            this.showElement(modalButton)

            this.showElement(modalInput1)

            modalButton.textContent = methodAction

        } *else* {

            this.hideElement(modalButton)

            this.hideElement(modalInput1)

            const result = this.linkedList[*methodName*]();

            const isBool = typeof result === 'boolean';

            let text = result;

*if* (isBool) {

                text = result ? 'Да' : 'Нет';

            }

            modalHeader1.textContent += ` ${text}`;

        }

        modalButton.onclick = () => {

            const inputValue1 = modalInput1.value;

            const inputValue2 = methodHeader2 ? modalInput2.value : undefined;

*if* (inputValue1 === '' || methodHeader2 && inputValue2 === '') {

*return*;

            }

*if* (methodHeader2 === undefined) {

*if* (*methodsOptions*.needAnswer){

                    const res = this.linkedList[*methodName*](inputValue1);

*if* (res !== undefined) {

                        this.showElement(modalHeader2);

                        modalHeader2.textContent = *methodsOptions*.answer + res;

                        this.hideElement(modalButton)

                    }

                }

                this.linkedList[*methodName*](inputValue1);

            } *else* {

*if* (*methodsOptions*.needAnswer){

                    const res = this.linkedList[*methodName*](inputValue1, inputValue2);

*if* (res !== undefined) {

                        this.showElement(modalHeader2);

                        modalHeader2.textContent = *methodsOptions*.answer + res;

                        this.hideElement(modalButton)

                    }

                }

                this.linkedList[*methodName*](inputValue1, inputValue2);

            }

*if* (!*methodsOptions*.needAnswer) {

                this.hideElement(this.modalWindow);

                this.hideElement(modalHeader2);

            }

        }

    }

    hideElement = (*elem*) => {

*elem*.style.display = 'none';

    }

    showElement = (*elem*, *isFlex* = false) => {

*elem*.style.display = *isFlex* ? 'flex' : 'block';

    }

}

const a = new Main();

Остальной код лабораторной работы, отвечающих за верстку и стили страницы расположен на GitHub.