Отчет по лабораторной работе

По дисциплине «Структуры и алгоритмы обработки данных»

На тему:

«Методы поиска подстроки в строке»

Выполнил:

Студент группы

БСТ1902

Игнатов В.С.

Вариант №5

Москва 2021

Оглавление

Задание на лабораторную работу

Задание №1

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования

Алгоритмы:

1.Кнута-Морриса-Пратта

2.Упрощенный Бойера-Мура

Задание №2

Написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

Ход работы

Задание №1

Код программы

```
const searchSubstr = (elem, str, funcName, signature, register = false) => {
    if (!register) {
        elem = elem.toLowerCase();
        str = str.toLowerCase()
    }

    console.time(signature);
    console.log(funcName(str, elem, register));
    console.timeEnd(signature)
}

/* M + N сравнений в среднем */
const KMP = (text, word) => {
```

```
    const prefixFunc = (word) => {
        const prefixTable = [0];
        let prefixIndex = 0;
        let suffixIndex = 1;

        while (suffixIndex < word.length) {
            if (word[prefixIndex] === word[suffixIndex]) {
                prefixTable[suffixIndex] = prefixIndex + 1;
                suffixIndex += 1;
                prefixIndex += 1;
            } else if (prefixIndex === 0) {
                prefixTable[suffixIndex] = 0;
                suffixIndex += 1;
            } else {
                prefixIndex = prefixTable[prefixIndex - 1];
            }
        }

        return prefixTable;
    }

    if (word.length === 0) return 0;

    let textIndex = 0;
    let wordIndex = 0;

    const prefixArr = prefixFunc(word);

    while (textIndex < text.length) {
        if (text[textIndex] === word[wordIndex]) {
            if (wordIndex === word.length - 1) return (textIndex - word.length) +
 1

            wordIndex += 1;
            textIndex += 1;
        } else if (wordIndex > 0) {
            wordIndex = prefixArr[wordIndex - 1];
        } else {
            wordIndex = 0;
            textIndex += 1;
        }
    }

    return -1;
}

/* N/M в лучших случаях, M + N в среднем */
const BoyerMour = (str, substr) => {
    let strLen = str.length;
    let substrLen = substr.length;
```

```
    if (substrLen > strLen) return -1;

    const offsetTable = {};

    for (let i = 0; i <= 255; i++) {
        let char = String.fromCharCode(i);
        offsetTable[char] = substrLen;
    }

    for (let i = 0; i < substrLen - 1; i++) {
        offsetTable[substr.charAt(i)] = substrLen - i - 1;
    }

    let i = substrLen - 1,
        j = i,
        k = i;

    while (j >= 0 && i <= strLen - 1) {
        j = substrLen - 1;
        k = i;

        while (j >= 0 && str.charAt(k) === substr.charAt(j)) {
            k--;
            j--;
        }

        i += offsetTable[str.charAt(i)]
    }

    if (k >= strLen - substrLen) return -1;
    else return k + 1;
}

function searching() {
    const str = prompt("Введите строку");
    const substr = prompt("Введите подстроку");

    console.time("Стандартная функция")
    str.indexOf(substr);
    console.timeEnd("Стандартная функция")

    searchSubstr(substr, str, KMP, "Алгоритм Кнута-Морриса-Пратта");
    searchSubstr(substr, str, BoyerMour, "Алгоритм Бойера-Мура");
}

searching();
```

Результат

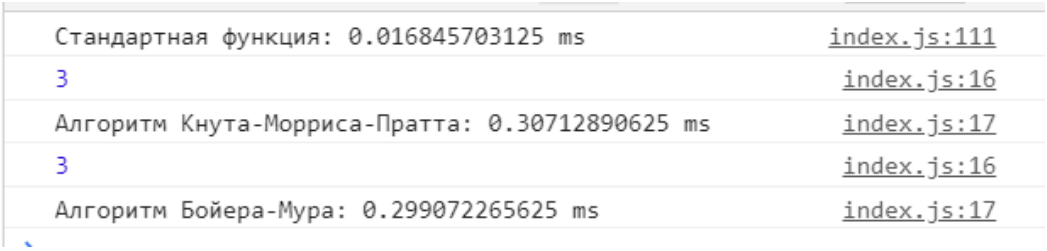

Рисунок 1 – Результат работы программы

Задание №2

Код программы

```
import PriorityQueue from './../../modules/PriorityQueue/PriorityQueue';

export default class Solver {
    constructor(table) {
        this.closed = new Map();
        this.opened = new Map();
        this.length = table.matrix.length;

        this.opened.set(table.getUnique(), table);
    }

    getMin() {
        return this.opened.get([...this.opened.keys()].reduce((curMin, key) => {
            const f = this.opened.get(key).f;
            if (f < curMin.min) {
                curMin.min = f;
                curMin.key = key;
            }
            return curMin;
        }, {
            min: +Infinity,
            key: null
        }).key);
    }

    isSolveable() {
        const curNode = this.getMin();
        const array = curNode.getUnique().split(',').map(val => +val);

        let n = curNode.zero.y + 1;
        array.forEach((value, index) => {
            if (value !== 0) {
                for (let i = index + 1; i < array.length; i++) {
```

```
                    if (array[i] < value && array[i] !== 0) {
                        n++;
                    }
                }
            }
        });
        return n % 2 === 0;
    }

    _getChain(solution, short = false) {
        const result = [];
        while (solution) {
            result.push(short ? (solution.zero.y * solution.dimension) + solution
.zero.x + 1 : solution.printPretty());
            solution = solution.parent;
        }
        return result.reverse();
    }

    search(short = false) {
        if (this.length === 4) {
            if (!this.isSolveable()) {
                return null;
            }
        }

        while (this.opened.size !== 0) {
            const curNode = this.getMin();
            // console.log(`opened: ${this.opened.size} closed: ${this.closed.siz
e} table: ${curNode.getUnique()}`);

            if (curNode.isSolve()) {
                return this._getChain(curNode, short);
            }

            this.opened.delete(curNode.getUnique());
            this.closed.set(curNode.getUnique(), curNode);

            const nextStages = curNode.nextStages();
            nextStages.forEach(table => {
                if (this.closed.has(table.getUnique())) {
                    return;
                }
                const repeat = this.opened.get(table.getUnique());
                if (!repeat) {
                    this.opened.set(table.getUnique(), table);
                } else {
                    if (table.g < repeat.g) {
                        this.opened.set(table.getUnique(), table);
                    }
                }
```

```
            });
        }

        return null;
    }
}
```

Результат

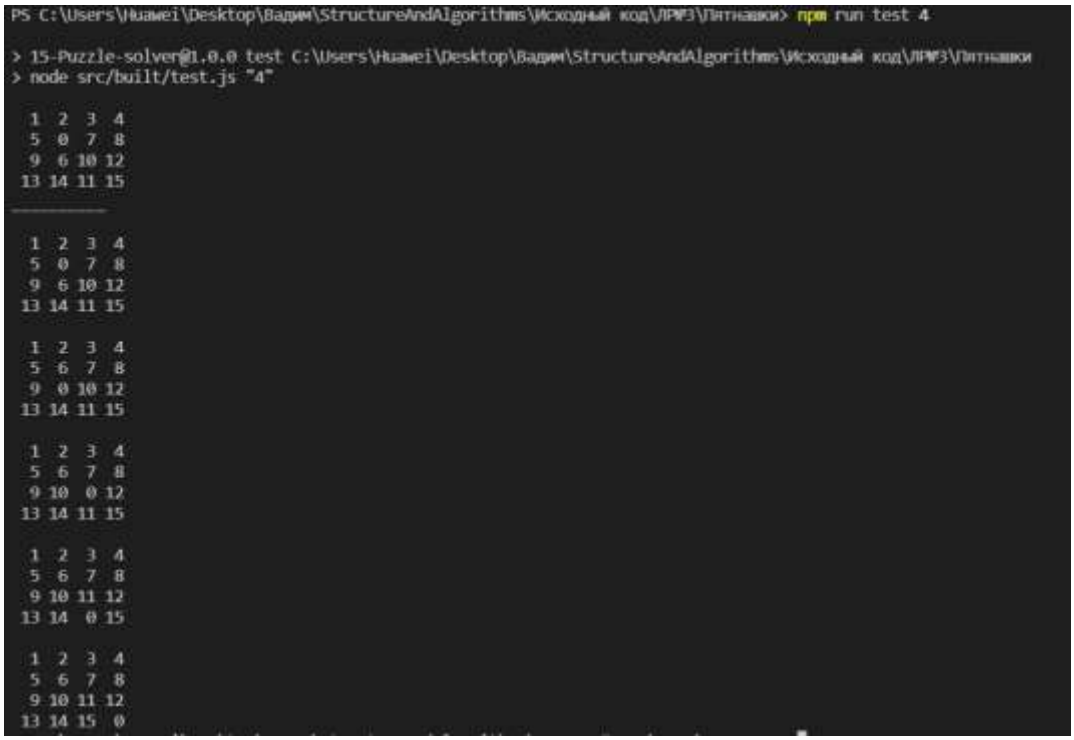

Рисунок 2 – Результат работы программы

Вывод

Я научился работать с алгоритмами поиска подстроки в строке и алгоритмом A*.