МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное образовательное учреждение высшего образования

САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С. Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1  Сортировки. Определение сложности алгоритма |
|  |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4321 |  |  |  | К.А. Лебедев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург, 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc178416272)

[2 Задание 4](#_Toc178416273)

[3 Ход разработки 5](#_Toc178416274)

[4 Исходный код программы 6](#_Toc178416275)

[5 Результаты работы программы 11](#_Toc178416276)

[6 Вывод 12](#_Toc178416277)

**1 Цель работы**

Изучить работу алгоритмов сортировок. Научиться определять сложность алгоритмов

**2 Задание**

Написать программу, которая будет парсить текст, написанный кириллицей, не учитывая цифры, сортировать методом Шелла по возрастания и записывать файлы с отсортированным текстом и анализом работы программы.

**3 Ход разработки**

Были реализованы класс задачи, сортировка Шелла, парсер регулярного выражения под задачу, написана утилита для прочтения чисел из командной строки на языке TypeScript, интегрирован компилятор в JavaScript, который сразу собирает TypeScript проект без предварительной компиляции.

**4 Исходный код программы**

*// index.ts*  
  
import Task from "./lib/Task";

new Task(); *// инициализация задачи*

*// utils/readline.ts*

import readline from "node:readline";

*// readline интерфейс*

export const readlineInterface = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

*// utils/shellSort.ts*

export const shellSort = <T *extends* string>(arr: T[]): T[] => {

*// Создаем копию массива, чтобы не изменять исходный*

const sortedArray = [...arr];

const n = sortedArray.length;

*// Устанавливаем интервал, начиная с половины длины массива*

for (

let interval = Math.floor(n / 2);

interval > 0;

interval = Math.floor(interval / 2)

) {

*// Сортируем элементы с использованием сортировки вставками в каждом цикле*

for (let i = interval; i < n; i++) {

let temp = sortedArray[i];

let j;

*// Вставляем элемент в отсортированную часть массива*

for (

j = i;

j >= interval &&

sortedArray[j - interval].localeCompare(temp) > 0;

j -= interval

) {

sortedArray[j] = sortedArray[j - interval];

}

sortedArray[j] = temp;

}

}

*// Возвращаем новый отсортированный массив*

return sortedArray;

};

*// utils/replaceMarks.ts*

const regex = /[^а-яА-ЯёЁ]/g;

export const replaceMarks = (value: string) => {

return value.replace(regex, "");

};

*// lib/Task.ts*

import { readlineInterface } from "../utils/readline";

import path from "path";

import fs from "fs";

import { replaceMarks } from "../utils/replaceMarks";

import { shellSort } from "../utils/shellSort";

class Task {

*private* filePath: string;

*private* fileString: string;

*private* file: string[];

*private* result: string;

*private* runTime: number;

*private* analyseData: string;

constructor() {

this.filePath = path.resolve(\_\_dirname); *// длина массива родителей*

this.fileString = ""; *// файл*

this.file = []; *// массив строк файла*

this.result = ""; *// результат сортировки*

this.analyseData = ""; *// результат анализа*

this.runTime = 0; *// время выполнения сортировки*

this.read(); *// инициализация таски*

}

*// парсинг файла*

parseFile() {

this.file = this.fileString

.split("\n")

.flatMap(item => item.split(" ").map(replaceMarks).filter(Boolean));

}

*// получение результата*

getResult() {

this.runTime = performance.now();

const sortedFile = shellSort(this.file);

this.runTime = Number((performance.now() - this.runTime).toFixed(0));

this.result = sortedFile.reduce((acc, item) => {

if (acc.length === 0) acc += item;

else if (acc[acc.length - 1][0] === item[0]) {

acc += ` ${item}`;

} else {

acc += `\n${item}`;

}

return acc;

}, "");

}

*// создание словаря кириллики*

getCreateCyrillicDictary(value: string) {

const cyrillicChars = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

*// Инициализируем объект для хранения символов и их количества*

const charDict: Record<string, number> = {};

*// Заполняем объект символами кириллицы с нулевыми значениями*

for (const char of cyrillicChars) {

charDict[char.toLowerCase()] = 0;

}

*// Подсчитываем вхождения символов из входной строки*

for (const char of value) {

if (charDict.hasOwnProperty(char)) {

charDict[char.toLowerCase()] += 1;

}

}

return charDict;

}

*// создание словаря символов*

getLetterDictary() {

this.analyseData += JSON.stringify(

this.getCreateCyrillicDictary(

this.result

.split("\n")

.flatMap(item => item.split(" "))

.join("")

.replace(" ", "")

),

null,

2

)

.replace(/^\{/, "")

.replace(/\}$/, "");

}

*// анализ*

analyse() {

this.analyseData += `Исходный текст: \n${this.fileString}\n`;

this.analyseData +=

"Параметры задания: кириллица, по алфавиту, по возрастанию, игнорировать числа, сортировка Шелла\n";

this.analyseData += `Количество слов: ${this.result.length}\n`;

this.analyseData += `Время выполнения: ${this.runTime} мс\n`;

this.analyseData += `Статистика: \n`;

this.getLetterDictary();

console.log(this.analyseData);

fs.writeFile(path.resolve("analysis.txt"), this.analyseData, () =>

console.log("Файл analysis.txt записан")

);

}

*// инициализация таски*

read() {

readlineInterface.question("", (query: string) => {

this.filePath = path.resolve(\_\_dirname, query);

fs.readFile(this.filePath, "utf-8", (err, data) => {

if (err) {

console.log("Файл не существует");

} else {

this.fileString = data;

this.parseFile();

this.getResult();

fs.writeFile(path.resolve("result.txt"), this.result, () =>

console.log("Файл result.txt записан")

);

this.analyse();

}

});

readlineInterface.close();

});

}

}

export default Task;

**5 Результаты работы программы**

Результат работы программы включает в себя вывод результат анализа программы

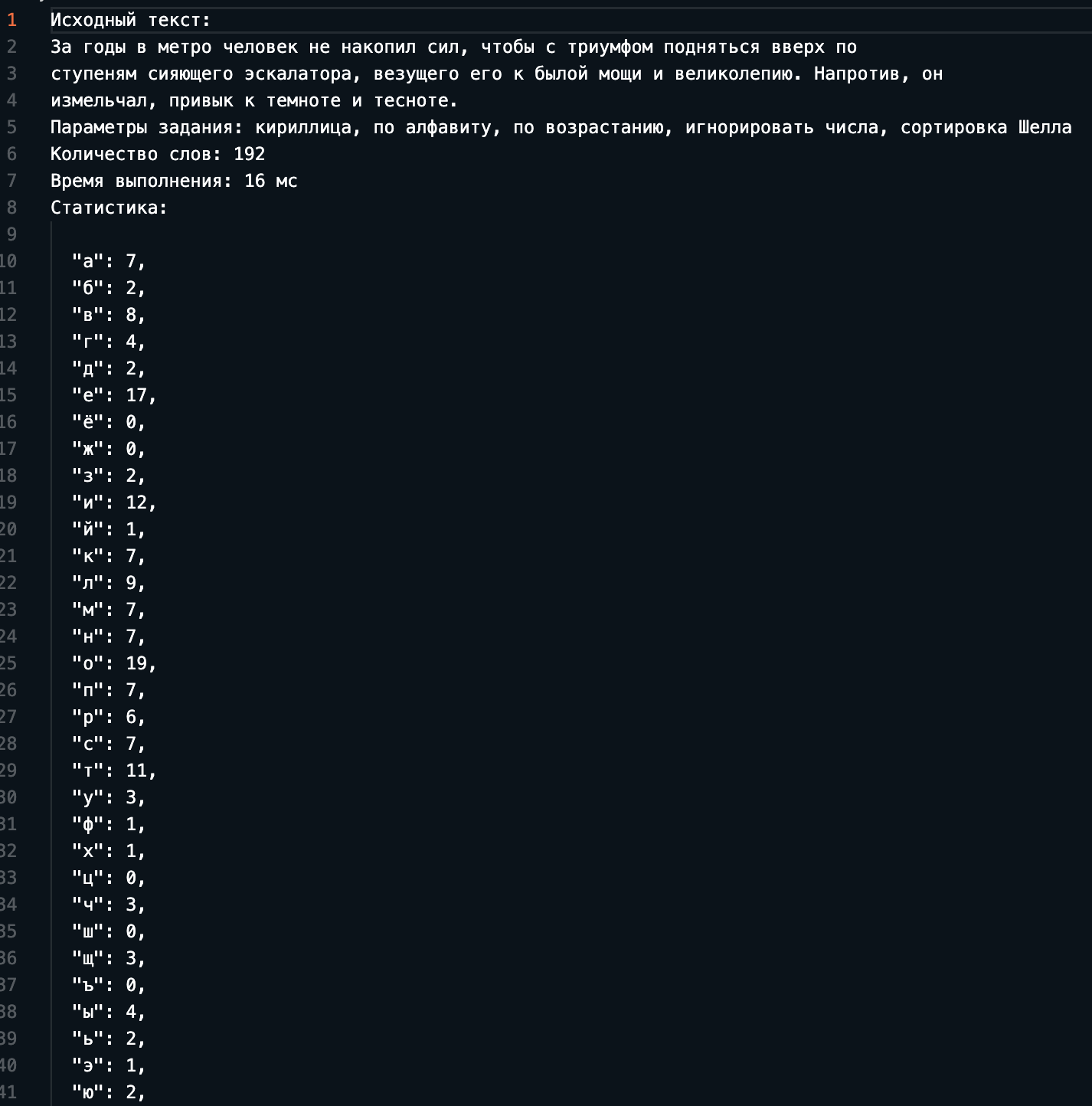


Рисунок 1 – тест программы №1

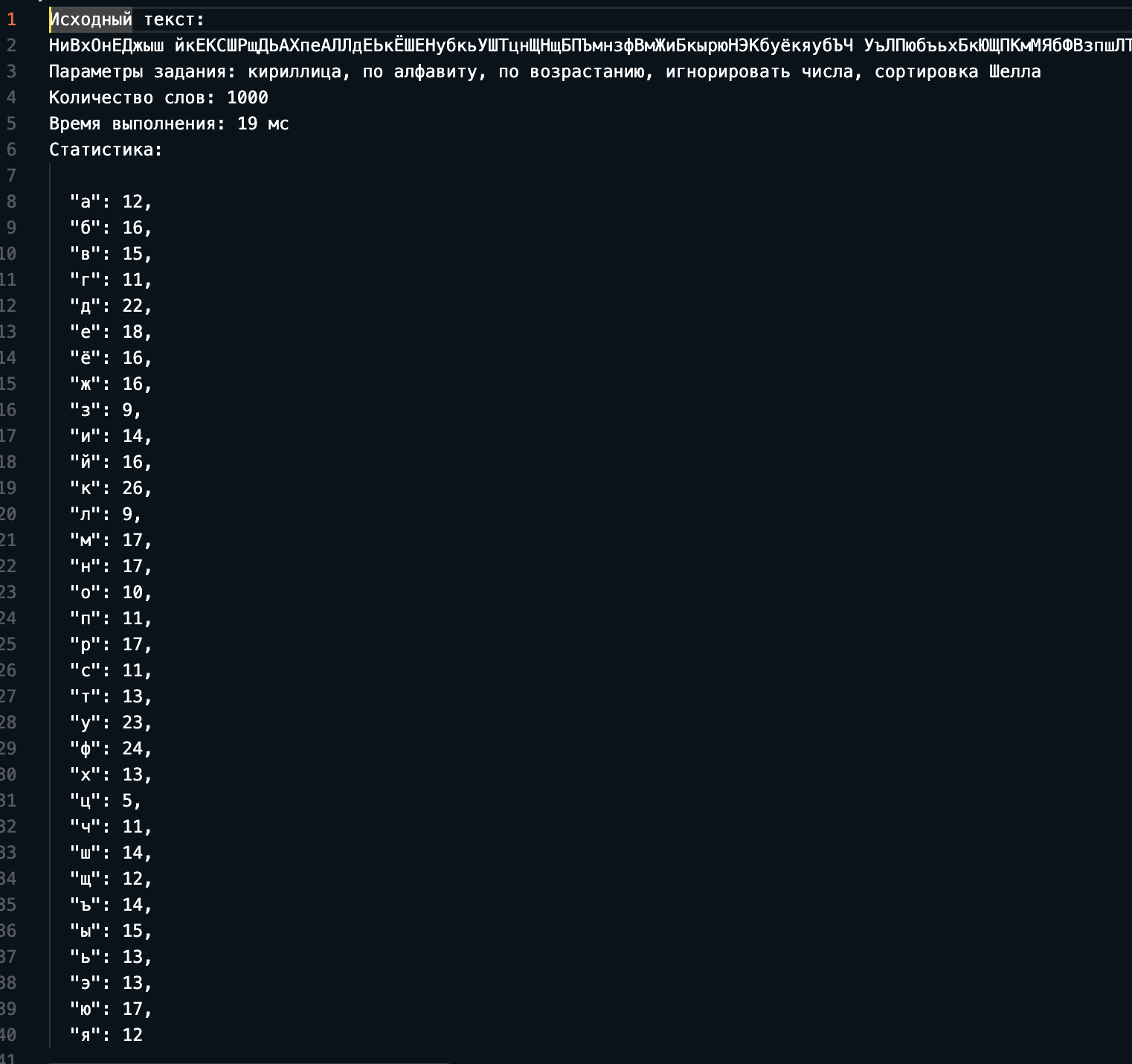


Рисунок 2 – тест программы №2

**6 Вывод**



Рисунок 3 – Диаграмма результатов программы

На рисунке 3 изображена диаграмма результатов в соответствии с процессорным временем выполнения программы сортировки Шелла.

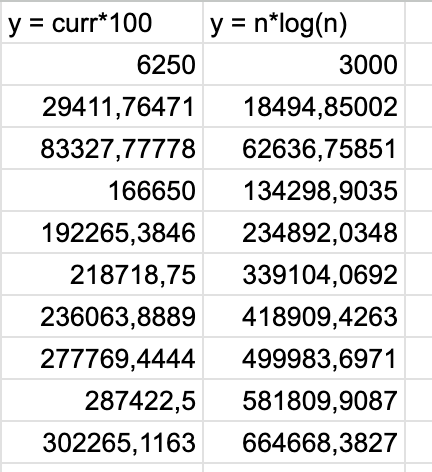


Рисунок 4 – Таблица вычислений

На рисунке 4 изображена таблица зависимости процессорного времени выполнения и теоретического времени выполнения сортировки Шелла. В данной таблице процессорное время было умножено на коэффициент 100, в соответствии с теоретическим коэффициентом превосходства мощности процессора. Исходя из описанных выше данных можно сделать вывод о том, что сложность алгоритма сортировки Шелла в лучшем случае по нотации BigO принадлежит к линейно-логарифмической сложности O(N\*log(N)), а в худшем к квадратичной сложности O(N^2). Можно заметить, что зачастую процессорное время выполнения не совпадает с теоретическим из-за продвинутости технологий. Хотелось бы отметить, что сортировка Шелла не является эффективным алгоритмом сортировки, поэтому зачастую, в продуктовой разработке при работе с трудоёмкими вычислениями используют сортировку слиянием, которая принадлежит к линейно-логарифмической сложности O(N\*log(N)).