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 №5  Максимум в скользящем окне |
|  |
| 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**

Найти максимум в каждом окне размера m данного массива чисел A[1 ... n]

**2 Задание**

Написать программу, которая найдет максимумы массива. Необходимо реализовать программу, которая будет считывать количество длину массива, массив, длину окна и находить максимумы в каждом окне в O(N).

**3 Ход разработки**

Были реализованы классы очереди и задачи, 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,

});

*// interfaces/interfaces.ts*

*// Дженерик очереди*

export interface IQueue<T> {

value: T[];

push: (item: T) => void;

shift: () => T | undefined;

}

*// lib/Queue.ts*

import { IQueue } from "../interfaces/interfaces";

*// Класс очереди*

class Queue<T> *implements* IQueue<T> {

*public* value: T[]; *// значение очереди*

constructor(arr: T[]) {

this.value = arr; *// создаем массив*

}

*// добавление элемента в очередь*

push(item: T) {

this.value.push(item);

}

*// удаление элемента из очереди*

shift() {

return this.value.shift();

}

}

export default Queue;

*// lib/Task.ts*

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

import Queue from "./Queue";

*// Класс Задачи*

class Task {

*private* res: number[];

*private* window: Queue<number>;

*private* resLength: number;

*private* windowLength: number;

*private* questionRes: number[];

constructor() {

this.init();

this.res = []; *// входящий массив*

this.resLength = 0; *// длина входящего массива*

this.windowLength = 0; *// длина окна*

this.window = new Queue([]); *// окно (очередь)*

this.questionRes = []; *// массив максимумов*

}

*// вычисление максимумов в O(N)*

calculateTask() {

this.window = new Queue(this.res.slice(0, this.windowLength)); *// инициализация окна в первоначальное состояние*

this.questionRes.push(Math.max(...this.window.value)); *// добавление в массив максимумов*

for (let i = this.windowLength; i < this.resLength; i++) {

this.window.push(this.res[i]); *// First Input*

this.window.shift(); *// First Output*

this.questionRes.push(Math.max(...this.window.value)); *// добавление в массив максимумов*

}

console.log(this.questionRes.join(" ")); *// вывод результата*

}

*// чтение данных*

readTask() {

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

this.resLength = parseInt(length); *// парсинг длины массивов*

if (isNaN(this.resLength) || this.resLength <= 0) {

console.log(

"Пожалуйста, введите корректное положительное число."

);

readlineInterface.close();

return;

}

readlineInterface.question("", input => {

const res = input.split(" ").map(Number); *// парсинг входящего массива данных*

if (res.length !== this.resLength || res.includes(NaN)) {

console.log(

"Пожалуйста, введите корректное положительное число."

);

readlineInterface.close();

return;

} else {

this.res = res;

readlineInterface.question("", windowLength => {

this.windowLength = parseInt(windowLength); *// парсинг длины окна*

if (

isNaN(this.windowLength) ||

this.windowLength <= 0 ||

this.windowLength > this.resLength

) {

console.log(

"Пожалуйста, введите корректное положительное число."

);

readlineInterface.close();

return;

}

this.calculateTask(); *// вычисление по задаче*

readlineInterface.close();

});

}

});

});

}

*// инициализация задачи*

init() {

this.readTask();

}

}

export default Task;

**5 Результаты работы программы**

Результат работы программы включает в себя вывод очереди команд



Рисунок 1 – тест программы №1



Рисунок 2 – тест программы №2



Рисунок 3 – тест программы №3

**6 Вывод**

Алгоритмы и структуры данных играют ключевую роль в эффективном решении задач в программировании. Они позволяют оптимизировать процессы и ускорить выполнение операций при правильном использовании.

Очередь, как одна из основных структур данных, реализует принцип FIFO (First In, First Out), что подразумевает, что первый элемент, добавленный в очередь, будет первым, который будет извлечён. Эта структура находит широкое применение в различных областях, включая управление задачами, обработку данных и сетевые технологии.

В ходе лабораторной работы был изучен механизм работы очереди, реализованы утилитарные функции для обработки команд, что позволило глубже понять её применение и преимущества в программировании. Это исследование подчеркнуло важность выбора правильной структуры данных для достижения максимальной эффективности в решении конкретных задач.