ГУАП

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. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Цель работы 3
2 Задание 3
3 Краткое описание хода разработки 3
4 Исходный код программы 4
5 Результаты работы программы с примерами 5
6 Выводы 6

1 Цель работы

Целью данной лабораторной работы является изучение алгоритмов оптимизации с использованием структур данных, таких как двусторонняя очередь (deque), для эффективного нахождения максимумов в скользящем окне заданного массива чисел.

2 Задание

По заданию было необходимо реализовать алгоритм поиска максимумов в каждом скользящем окне размера m массива чисел A[1…n] со временем работы O(n).

3 Краткое описание хода разработки

1. Для достижения оптимальной сложности O(n) решено использовать двустороннюю очередь (deque). Deque позволяет эффективно добавлять и удалять элементы с обеих сторон. В данной задаче используется для хранения индексов элементов массива, которые потенциально могут быть максимумами в текущем окне.
2. На каждом шаге обработка текущего элемента выполняется в 3 этапа:
   - Удаление из deque элементов, выходящих за пределы текущего окна.
   - Удаление элементов, меньших текущего, так как они не могут быть максимумами.
   - Добавление текущего индекса в deque.

   Как только окно полностью заполнено ($i \geq m-1$), добавляется максимум текущего окна (первый элемент в deque) в список результатов.
3. Программа протестирована на нескольких примерах, включая крайние случаи, такие как:
   - Окно размера 1 (m=1), где максимум каждого окна — сам элемент.

– Окно размера n, где результатом является максимум всего массива.

– Большие массивы для проверки производительности.

4 Исходный код программы

```
from collections import deque


# Находит максимум в каждом окне размера m массива array.
def find_max_sliding_window(n, array, m):
    # Дек для хранения индексов элементов текущего окна
    deque_indices = deque()
    max_in_windows = []

    for i in range(n):
        # Удаляем элементы, которые выходят за пределы окна
        if deque_indices and deque_indices[0] < i - m + 1:
            deque_indices.popleft()

        # Удаляем из дека все элементы, которые меньше текущего
        # Так как они не могут быть максимумом в текущем или последующих окнах
        while deque_indices and array[deque_indices[-1]] < array[i]:
            deque_indices.pop()

        # Добавляем текущий индекс в дек
        deque_indices.append(i)

        # Максимум окна находится в начале дека
        # Начинаем добавлять максимумы в список только после того, как
заполнили первое окно
        if i >= m - 1:
            max_in_windows.append(array[deque_indices[0]])

    return max_in_windows


def main():
    # Чтение входных данных
    n = int(input("Введите n: ").strip())
    array = list(map(int, input("Введите массив: ").strip().split()))
    m = int(input("Введите m: ").strip())

    # Вычисление максимумов в каждом окне
    results = find_max_sliding_window(n, array, m)

    # Вывод результата
    print(" ".join(map(str, results)))


if __name__ == "__main__":
    main()
```

5 Результаты работы программы с примерами

На рис. 1, 2, 3 изображено тестирование написанной программы с различными входными данными.

```
Введите n: 8
Введите массив: 2 7 3 1 5 2 6 2
Введите m: 4
7 7 5 6 6

Process finished with exit code 0
```

Рисунок 1

```
Введите n: 12
Введите массив: 1 2 3 4 5 6 7 8 9 10 11 12
Введите m: 2
2 3 4 5 6 7 8 9 10 11 12

Process finished with exit code 0
```

Рисунок 2

```
Введите n: 16
Введите массив: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1
Введите m: 5
5 5 5 5 5 5 5 5 5 5 5 5

Process finished with exit code 0
```

Рисунок 3

6 Выводы

- В результате выполнения лабораторной работы было изучено использование двусторонней очереди (deque) для оптимизации вычислений в задачах, связанных с окнами фиксированного размера.
- Разработан алгоритм нахождения максимумов в скользящем окне со сложностью O(n), что позволило эффективно обрабатывать массивы длиной до 10^5.
- Работа продемонстрировала важность выбора подходящей структуры данных для оптимизации алгоритмов, что является ключевым навыком в разработке высокопроизводительных приложений.