

Санкт-Петербургский Национальный Исследовательский Университет  
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий и систем связи

**Лабораторная работа №5**

**Вариант №2**

Выполнил(и:)

Алексеев Т.

Бабаев Р.

Проверил

Мусаев А.А.

Санкт-Петербург,

2023

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1. ЗАДАНИЕ 1.....	4
2. ЗАДАНИЕ 2.....	5
3. ЗАДАНИЕ 3.....	6
4. ЗАДАНИЕ 4.....	7
5. ЗАДАНИЕ 5.....	8
6. ЗАДАНИЕ 6.....	9
7. ЗАДАНИЕ 7.....	10
ЗАКЛЮЧЕНИЕ.....	11
СПИСОК ЛИТЕРАТУРЫ .....	12

## ВВЕДЕНИЕ

Целью данной работы являлось повторение всего материала, пройденного за два семестра.

Для достижения данной цели необходимо было выполнить следующие задания:

1. Задание 1: Разработайте алгоритм, проверяющий результат игры в крестики-нолики (3x3).
2. Задание 2: Для заданной матрицы  $M \times N$ , в которой каждая строка и столбец отсортированы по возрастанию, напишите метод поиска элемента.
3. Задание 3: Напишите алгоритм, находящий все варианты расстановки восьми ферзей на шахматной доске размером 8x8 так, чтобы никакие две фигуры не располагались на одной горизонтали, вертикали или диагонали.
4. Задание 4: Ребёнок поднимается по лестнице из  $n$  ступенек. За один шаг он может переместиться на одну, две или три ступеньки. Реализуйте метод, рассчитывающий количество возможных вариантов перемещения ребёнка по лестнице.
5. Задание 5: Опишите, как бы вы использовали один одномерный массив для реализации трёх стеков.
6. Задание 6: Напишите максимально короткий код для экспоненциального фильтра.
7. Задание 7: Дан неотсортированный массив целых чисел. Верните наименьшее пропущенное целое число. Алгоритм должен выполняться за время  $O(n)$ .

Решения данных задач будут находиться на GitHub по ссылке:  
[https://github.com/NorthPole0499/Algoritms\\_task\\_10](https://github.com/NorthPole0499/Algoritms_task_10)

## 1. ЗАДАНИЕ 1

Для проверки результата игры пользователю необходимо ввести итоговое поле построчно. Далее происходит проверка на необходимую правильность ввода и на допустимое количество крестиков и ноликов.

Если данные условия выполняются, то происходит проверка результатов игры. Проверяются одинаковость символов на каждой строке, в каждом столбике и на двух диагоналях. Если какое-то из условий выполняется, то вызывается функция `ans_generator()`, которая выводит конкретного победителя, крестики или нолики. Иначе, выводится сообщение о ничье.

```
1 - крестики, 0 - нолики
Введите законченное поле в строку ввода по одной строки.
Значки вводите через пробел, после каждых трёх нажмите кнопку Enter
1 0 1
0 1 0
1 0 1
Победили крестики!
```

Приложение 1 – Пример вывода программы для Задания 1

## 2. ЗАДАНИЕ 2

В самом начале пользователю предлагается ввести размер матрицы, а далее ввести элементы этой матрицы построчно. Также, указывается искомый элемент.

Так как строки матрицы отсортированы по возрастанию, то проходимся по каждой из них циклом `for`. Если искомое число лежит в диапазоне между первым и последним элементом данной строки, то есть смысл искать в этой строке элемент. И вызывается функция `bin_find()`, которая реализует бинарный поиск по данной строке. Это возможно сделать благодаря изначальной сортировки по возрастанию. Если элемент найден, то переменная `flag` переводится в ложное значение и заканчивается цикл. Если же элемент не найден, то цикл `for` совершает следующую итерацию. Если же по истечению всех итераций элемент не будет найден, то выводятся координаты `(-1, -1)`.

```
Введите размеры матрицы MxN через пробел: 3 3
Введите элементы 1 строки матрицы по возрастанию через пробел: 1 2 3
Введите элементы 2 строки матрицы по возрастанию через пробел: 4 5 6
Введите элементы 3 строки матрицы по возрастанию через пробел: 7 8 9
Введите искомый элемент матрицы: 8
3 2
```

Приложение 2 – Пример вывода программы для Задания 2

### 3. ЗАДАНИЕ 3

Задача о восьми ферзях – классическая комбинаторная задача. Мы решили её динамическим перебором возможных вариантов расположения ферзей. Изначально рассматривается 8 вариантов расположения ферзей на 1 строке. Затем для каждого из этих вариантов ферзи расставляются на 2 строке, затем всевозможные варианты расположения на 3 строке и т.д. Для удобства была реализована специальная функция `under_attack(col, queens)` – она сравнивает текущее возможное положение ферзя с уже стоящими и возвращает значение `True` или `False` в зависимости от того, находится он под атакой или нет.

Таким образом, было получено 92 возможные расстановки, что соответствует теоретическим расчетам.

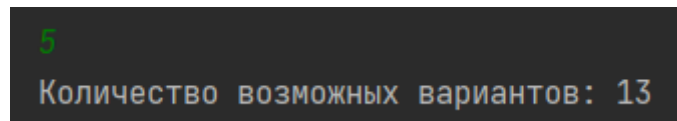
```
[7 4 2 5 8 1 3 6]
[7 4 2 8 6 1 3 5]
[7 5 3 1 6 8 2 4]
[8 2 4 1 7 5 3 6]
[8 2 5 3 1 7 4 6]
[8 3 1 6 2 5 7 4]
[8 4 1 3 6 2 7 5]
Количество возможных расстановок: 92
```

Приложение 3 – Пример вывода программы для Задания 3

#### 4. ЗАДАНИЕ 4

Данная задача является отличным примером для решения методом динамического программирования. Так она и была решена, с помощью написания функции и рекурсии.

Сначала пользователь вводил  $n$  - количество ступенек. И сразу же после этого запускалась функция `walk()`, на вход которой подавалось  $s$  - текущий номер ступеньки, на которой стоит ребёнок. Логика функции такая: если текущий номер ступеньки больше, чем  $n$ , то возвращается ноль. Если текущий номер равен общему количеству ступенек, то возвращается 1. Иначе же, возвращается сумма рекурсий, где вызывается функция `walk()` и номер текущий ступеньки увеличивается на 1, 2 и 3. Таким образом, в конце функция возвращает количество возможных вариантов перемещения.



```
5
Количество возможных вариантов: 13
```

Приложение 4 – Пример вывода программы для Задания 4

## 5. ЗАДАНИЕ 5

Для более лёгкого описания выстраивания трех стеков в одномерном массиве был написан класс `ThreeStack`, имеющий методы для каждого из трёх стеков. При инициализации создаётся одномерный массив из трёх нулей – границ каждого стека, условного их дна. Также, записывается переменная `index`, в который записан индекс границы второго стека.

Первый стек будет располагаться в начале массива. Метод `get_1()` возвращает верхний элемент в 1 стеке, то есть возвращает элемент в массиве с индексом 0. Метод `add_1()` добавляет в начало массива заданный элемент.

Второй стек будет располагаться в середине массива. Метод `get_2()` возвращает верхний элемент во 2 стеке, то есть возвращает элемент в массиве с индексом `index - 1`, ближайший к границе. Метод `add_2()` добавляет элемент в массив с помощью `insert` на место `index - 1`.

Третий стек будет располагаться в конце массива. Метод `get_3()` возвращает верхний элемент в 3 стеке, то есть возвращает элемент в массиве с индексом -1. Метод `add_1()` добавляет в конец массива заданный элемент.

Также, в коде прописаны проверки на пустоту массива при вызовах методов `get_#()`. Для лучшего понимания задуманной реализации был написан код, загруженный на GitHub.

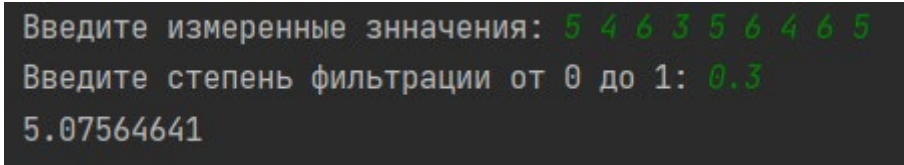


## 6. ЗАДАНИЕ 6

В данном задании было необходимо реализовать максимально короткий код для экспоненциального фильтра.

2 строки были потрачены на реализацию ввода измеренных значений и степени фильтрации. Нами была написана функция `exp_filter()`, которая принимала три значения: массив с измеренными значениями, предыдущее фильтрованное значение и степень фильтрации. Для сокращения количества строк была выбрана рекуррентная формула и реализована рекурсия. В итоге функция заняла 5 строк кода. И была написана строчка вывода. Также, одна строка была оставлена пустой между функцией и `main`-ом для лучшего понимания кода.

Таким образом, реализация экспоненциального фильтра заняла у меня 9 строчек.



```
Введите измеренные значения: 5 4 6 3 5 6 4 6 5
Введите степень фильтрации от 0 до 1: 0.3
5.07564641
```

Приложение 5 – Пример вывода программы для Задания 6

## 7. ЗАДАНИЕ 7

В данной задаче необходимо найти минимальный пропущенный элемент в массиве за  $O(n)$ . Для этого массив случайной длины (от 10 до 20) заполняется случайными числами (от 0 до 100).

После этого находятся минимальный и максимальный элементы. Это необходимо для создания массива с булевыми переменными `flags` длиной максимальный элемент – минимальный. Сначала он заполняется значениями `False`.

Далее циклом `for` идём по имеющемуся массиву с числами и отмечаем в массиве `flags` советующий ему индекс, меняем на `True`. После этого остается лишь одно действие: опять начать цикл `for` по массиву `flags` до первого элемента со значением `False`. Как только это произошло, число, соответствующее данному индексу, выводится в консоль, цикл прерывается, и программа заканчивает свою работу.

Таким образом, если говорить грубо, то программа совершает работу за  $O(5n)$ . Но зная, что константы-множители отбрасываются при измерении сложности, можно сказать, что мы выполнили условие и данный алгоритм выполняется за  $O(n)$ .

```
Имеющийся массив: [2, 67, 66, 4, 6, 7, 72, 99, 46, 79, 15, 49, 82, 83, 53, 55, 23, 28]  
Минимальное пропущенное число: 3
```

Приложение 6 – Пример вывода программы для Задания 7

## **ЗАКЛЮЧЕНИЕ**

Таким образом, в ходе выполнения данной лабораторной работы были достигнуты все поставленные цели. Произошло повторение всей программы, которую удалось пройти за два семестра по данному предмету.

## СПИСОК ЛИТЕРАТУРЫ

- 1) Всё о сортировке в Python: исчерпывающий гайд – Tproger // URL:  
<https://tproger.ru/translations/python-sorting/>  
(дата обращения: 29.05.2023)
- 2) Экспоненциальный фильтр // URL:  
[http://www.pmg.org.ru/asutp/scada\\_filter.pdf](http://www.pmg.org.ru/asutp/scada_filter.pdf) (дата обращения:  
29.05.2023)