Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Лабораторная работа №1

Вариант №1

Выполнили:

Волгин Л.А.

Шляхов Д.О.

Проверил:

Мусаев А.А.

Санкт-Петербург,

2022

!!!*Код всех программ можно просмотреть, перейдя по приложенной ссылке на гитхаб.!!!
https://github.com/Puroblast/Programming_hoomework

Задание №1

Описание задания:

Реализовать алгоритм фасетного поиска для определенной группы объектов (Таблица 1). Реализовать алгоритм не менее чем для 16 объектов. Пользователь может отвечать на вопросы только «да» и «нет».

Пример:

Дано: 1, 2, 32, 13.

-Вы загадали цифру, но не число?

-Да.

-Оно является четным?

-Да.

Ответ: 2

Листинг (Решение):

Для решения поставленной задачи мы решили использовать структуру сложного словаря, состоящего из других словарей, в последнем из которых хранится название животного (Рисунок 1).

```
animals = {
    "травоядное?": {
        "рогатое?": {
            "парнокопытное?": {
                "стадное?": "Корова",
                "не стадное?": "Лось"
            },
            "не парнокопытное?": {
                "стадное?": "Слон",
                "не стадное?": "Носорог"
            }
        },
        "не рогатое?": {
            "парнокопытное?": {
                "стадное?": "Бегемот",
                "не стадное?": "Кабарга"
            },
            "не парнокопытное?": {
                "стадное?": "Лошадь",
                "не стадное?": "Тапир"
            }
        }
```

Рисунок 1. Фрагмент словаря, хранящего характеристики животных.

Далее пользователю показывается список всех животных, содержащихся в словаре, для того, чтобы он загадал одно из них. Так как мы можем изменять словарь вручную, мы извлекаем список животных из данных словаря, а не просто храним статичный список, который никак не зависит от того, что содержится в словаре (Рисунок 2).

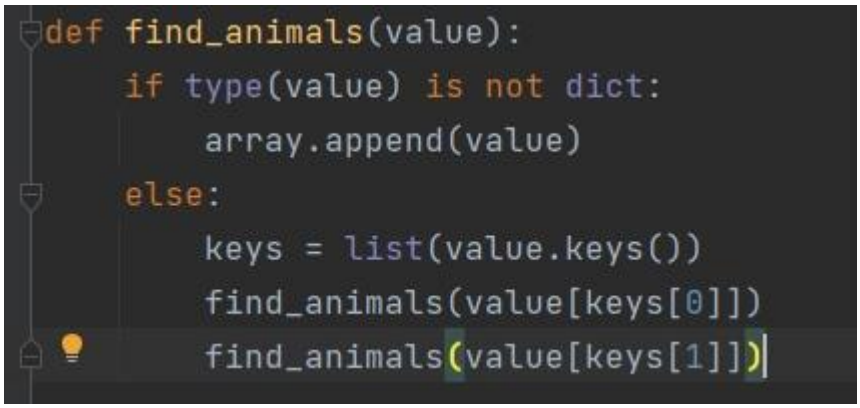

Рисунок 2. Функция, извлекающая названия всех животных из словаря.

Далее, с помощью функции question (Рисунок 3), вызывающей себя рекурсивно, до тех пор, пока переданное в неё значение не окажется строкой, происходит перемещение по словарю, в зависимости от ответов пользователя, и задаются наводящие вопросы. Когда программа доходит до конца словаря, пользователю выводится название загаданного им животного.

Рисунок 3. Функция question – основная часть программы.

Результат выполнения программы:

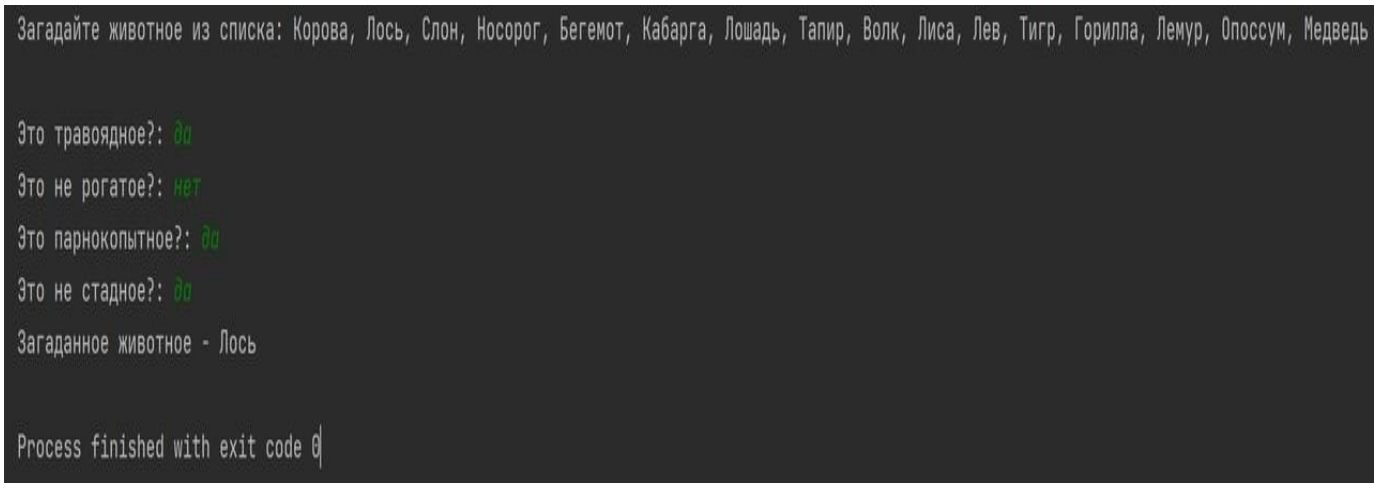

Рисунок 4. Результат выполнения программы.

Вывод:

Реализован алгоритм фасетного поиска, с помощью рекурсивной функции, пробегающейся по сложному словарю, содержащему вложенные словари.

Задание №2

Описание задания:

Пользователь задает любое количество чисел с экрана, разделяя их запятыми. Реализовать алгоритм, который распределяет числа на натуральные, целые, рациональные, вещественные, комплексные, четные, нечетные и простые. Обратите внимание, что цифры могут попадать в несколько категорий.

Листинг (Решение):

С помощью условных операторов выбирается нужная категория числа. Было реализовано несколько функций, для определения иррационального числа (Рисунок 5),

```
def irrational_check(number):
    splitted_number = number.split("**")
    degree_value = 1
    try:
        splitted_number = list(map(float, splitted_number))
        for i in range(len(splitted_number) - 1, 0, -1):
            degree_value = splitted_number[i] ** degree_value
        if float(degree_value).is_integer():
            full_check(str(splitted_number[0] ** degree_value))
        else:
            extended_numbers.append((splitted_number[0] ** degree_value))
            complex_numbers.append(str(complex(splitted_number[0] ** degree_value)).lstrip("(").rstrip(
                ")"))  # Приближенное значение иррационального числа
    except ValueError:
        print(f"Неверная запись числа: {number}")
```

Рисунок 5. Проверка числа на иррациональность.

А так же функция для определения простых, чётных, нечётных и натуральных чисел (Рисунок 6). Комплексные числа, не содержащие иррациональные коэффициенты, определяются с помощью встроенной функции complex().

```
def common_odd_even_natural(number):
    if number == 2:
        simple_numbers.append(number)
        even_numbers.append(number)
        natural_numbers.append(number)
    elif number % 2 == 0:
        even_numbers.append(number)
        if number > 0:
            natural_numbers.append(number)
    else:
        odd_numbers.append(number)
        if number > 0:
            natural_numbers.append(number)
            flag = True
            for j in range(3, int(number ** 1 / 2) + 1, 2):
                if number % j == 0:
                    flag = False
                    break
            if flag:
                simple_numbers.append(number)
```

Рисунок 6. Проверка числа на чётность/нечётность, простоту и натуральность.

Результат выполнения программы:

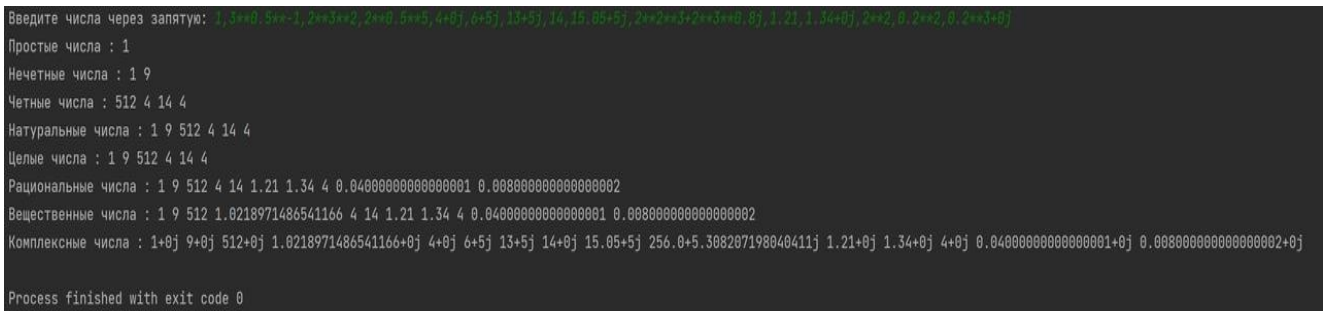

Рисунок 7. Результат выполнения программы.

Вывод:

Написан алгоритм распределения чисел по категориям, с помощью условных операторов.

Задание №3

Описание задания:

Пользователь задает массив натуральных чисел. Реализовать для них алгоритмы сортировки следующими методами: пузырьковый, гномий, блочный, пирамидальный. Проанализировать достоинства и недостатки данных методов.

Листинг (Решение):

Программа предоставляет пользователю вводить массивы чисел, выбирать способы их сортировки и прекращать ввод по желанию пользователя.

Пузырьковая сортировка (перестановкой двух соседних элементов по величине)(Рисунок 8) реализована с помощью двух вложенных циклов(после каждого вложенного цикла определяется максимальное число из оставшихся неотсортированных чисел, с переносом на правильное место). Сложность такого алгоритма – O(n^2).

```
def bubblesort(array):
    for i in range(len(array)):
        flag = True
        for j in range(len(array) - 1 - i):
            if array[j] >= array[j + 1]:
                array[j], array[j + 1] = array[j + 1], array[j]
                flag = False
        if flag:
            break
```

Рисунок 8. Пузырьковая сортировка.

Гномья сортировка (Рисунок 9) похожа на пузырьковую сортировку, и имеет такую же сложность (O(n^2)), но при этом после перестановки двух соседних элементов алгоритм не продолжает итерироваться дальше по массиву, а начинает возвращается в начало массива, от элемента с которым произошла перестановка, до тех пор, пока элемент не будет больше предыдущего элемента .

```
def gnomesort(array):
    for i in range(len(array) - 1):
        k = i + 1
        l = i
        while k != 0:
            if array[l] > array[k]:
                array[l], array[k] = array[k], array[l]
                k -= 1
                l -= 1
            else:
                break
```

Рисунок 9. Гномья сортировка.

Блочная сортировка (Рисунок 10) хорошо работает на целых числах. Числа распределяются по ячейкам массива, которые представляют из себя подмассивы. В случае, если несколько чисел попадают в один подмассив, они сортируются внутри этой ячейки. После этого массив восстанавливается, извлекая значения из каждой ячейке, слева направо. Диапазон значений, которые могут попадать в i-ую ячейку, определяется по специальной формуле, представленной на изображении с кодом. Вообще, в крайнем случае, когда все числа попадают в одну ячейку, сложность такого алгоритма равна (O(n^2)), но при большом разбросе значений вероятность попадания двух и более значений в одну ячейку крайне мала, поэтому с большой натяжкой можно сказать, что в среднем сложность такого алгоритма – O(n).

```python
def bucketsort(array):
    bucket_array = []
    for i in range(len(array)):
        bucket_array.append([])
    max = array[0]
    for i in range(1, len(array)):
        if array[i] > max:
            max = array[i]
    for i in range(len(array)):
        position = (array[i] * len(array)) // (max + 1)
        if len(bucket_array[position]) == 0:
            bucket_array[position].append(array[i])
        else:
            for j in range(len(bucket_array[position])):
                if array[i] < bucket_array[position][j]:
                    bucket_array[position].insert(j, array[i])
                    break
                elif j == len(bucket_array[position]) - 1:
                    bucket_array[position].append(array[i])

    i = 0  # Отвечает за перебор массива array
    k = 0  # Отвечает за перебор массива bucket_array
    while i < len(array):
        for j in bucket_array[k]:
            array[i] = j
            i += 1
        k += 1
```

Рисунок 10. Блочная сортировка.

Пирамидальная сортировка (Рисунок 11) работает следующим образом: сначала массив приводится к виду правильной пирамиды (пирамиды, в которой значение родителя больше либо равно значению его детей), с помощью функции maxheap() (Рисунок 12). Таким образом в верхушку пирамиды ставится максимальное значение. Далее оно извлекается и ставится в конец массива, а на его место встает самый правый лист дерева. Это происходит в функции heapsort(). После этого, к пирамиде, без учета её верхушки, перемещённой в конец массива, применяется функция heapify() (Рисунок 13)(функцию maxheap применять уже не нужно, так как по сути вся пирамида имеет правильный вид, кроме её верхушки). Таким образом, после каждого вызова функции heapify() внутри функции heapsort() определяется наибольший элемент из оставшихся в пирамиде. Поэтому функция heapify() вызывается внутри функции heapsort() n раз. Функция heapify() рекурсивна, и в самом худшем случае она пробегается по всей глубине дерева, равной log(n), поэтому, общая сложность равна n*log(n).

```
def heapsort(array):
    global heap_size
    maxheap(array)
    for i in range(len(array) - 1, 0, -1):
        array[0], array[i] = array[i], array[0]
        heap_size = heap_size - 1
        heapify(array, 0)
```

Рисунок 11. Пирамидальная сортировка.

```
def maxheap(array):
    middle = (len(array) // 2) - 1
    for i in range(middle, -1, -1):
        heapify(array, i)
```

Рисунок 12. Функция maxheap().

```python
def heapify(array, index):
    if (index + 1) * 2 <= heap_size + 1:
        left = (index + 1) * 2 - 1
        right = (index + 1) * 2
        if right >= heap_size + 1:
            maximum = left
        elif array[left] >= array[right]:
            maximum = left
        else:
            maximum = right
        if array[index] < array[maximum]:
            array[index], array[maximum] = array[maximum], array[index]
            heapify(array, maximum)
```

Рисунок 13. Функция heapify().

```python
commands = ["heapsort", "bucketsort", "gnomesort", "bubblesort", "exit"]
while True:
    list_of_numbers = change()
    heap_size = len(list_of_numbers) - 1
    command = input(f"Введите способ сортировки, доступные способы : {', '.join(commands)}: ")
    if command == commands[0]:
        heapsort(list_of_numbers)
        print(list_of_numbers)
    elif command == commands[1]:
        bucketsort(list_of_numbers)
        print(list_of_numbers)
    elif command == commands[2]:
        gnomesort(list_of_numbers)
        print(list_of_numbers)
    elif command == commands[3]:
        bubblesort(list_of_numbers)
        print(list_of_numbers)
    elif command == commands[4]:
        break
    else:
        print("Неверный ввод")
```

Рисунок 14. Основной код программы.

Результат выполнения программы:

```
Введите числа, через пробел: 1 54 3 5 90 13 24 57
Введите способ сортировки, доступные способы : heapsort, bucketsort, gnomesort, bubblesort, exit: heapsort
[1, 3, 5, 13, 24, 54, 57, 90]
Введите числа, через пробел: 1 54 3 5 90 13 24 57
Введите способ сортировки, доступные способы : heapsort, bucketsort, gnomesort, bubblesort, exit: bucketsort
[1, 3, 5, 13, 24, 54, 57, 90]
Введите числа, через пробел: 1 54 3 5 90 13 24 57
Введите способ сортировки, доступные способы : heapsort, bucketsort, gnomesort, bubblesort, exit: gnomesort
[1, 3, 5, 13, 24, 54, 57, 90]
Введите числа, через пробел: 1 54 3 5 90 13 24 57
Введите способ сортировки, доступные способы : heapsort, bucketsort, gnomesort, bubblesort, exit: bubblesort
[1, 3, 5, 13, 24, 54, 57, 90]
Введите числа, через пробел: 1 54 3 5 90 13 24 57
Введите способ сортировки, доступные способы : heapsort, bucketsort, gnomesort, bubblesort, exit: exit
```

Рисунок 15. Результат выполнения программы.

Вывод:

Реализованы разные виды сортировок. У каждого из представленных алгоритмов сортировки есть свои достоинства и недостатки, и каждый из них имеет свою область применения.

Задание №4

Описание задания:

С **A** третьекурсниками часто происходит событие **B**. Зная вероятность **N**, что данное событие произойдет с ними за **C** дней (**N** вводится пользователем при запуске программы для каждого из **A**), определите вероятность того, что за **D** дней данная ситуация произойдет только с третьекурсником **E**. Реализовать алгоритм для решения данной задачи.

Листинг (Решение):

В решении данной задачи используются математические формулы из теории вероятности.

Сначала для каждого студента высчитывается вероятность того, что за один день с ним ничего не произойдёт. Далее с помощью этой вероятности высчитываем вероятность того, что это событие произойдёт с Андреем за D дней, а также вероятности того, что это событие не произойдет с Машей и Таней (по отдельности) за D дней. Чтобы найти вероятность того, что это событие произойдет за D дней только с Андреем, надо перемножить найденные вероятности (так как все три условия должны выполнятся одновременно) (Рисунок 16).

```
def lucky_man():
    not_prob_one_day_andrey = (1 - n_a) ** (1 / c)  # Вероятность того что с Андреем ничего не произойдет за 1 день
    prob_d_day = 1 - not_prob_one_day_andrey ** d  # Вероятность того что с Андреем произойдет событие за D дней

    not_prob_one_day_masha = (1 - n_m) ** (1 / c)
    not_prob_d_day_masha = not_prob_one_day_masha ** d  # Вероятность того что с Машей ничего не произойдет за D дней

    not_prob_one_day_tanya = (1 - n_t) ** (1 / c)
    not_prob_d_day_tanya = not_prob_one_day_tanya ** d  # Вероятность того что с Таней ничего не произойдет за D дней
    return prob_d_day * not_prob_d_day_tanya * not_prob_d_day_masha
```

Рисунок 16. Расчёт вероятностей.

Результат выполнения программы:

```
Введите число от 0 до 1 - Вероятность происхождения события с Андреем за 1 дней: 0.999
Введите число от 0 до 1 - Вероятность происхождения события с Машей за 1 дней: 0.00001
Введите число от 0 до 1 - Вероятность происхождения события с Таней за 1 дней: 0.0000001
Вероятность того, что за 101 дней молния ударит только Андрея = 0.9989804150797508
```

Рисунок 17. Результат выполнения программы.

Вывод:

Никакого алгоритма не используется, используются только формулы для подсчёта вероятностей.

Задание №5

Описание задания:

Реализовать алгоритм, заполняющий таблицу неповторяющимися координатами x и y. Количество координат n равно квадратному корню из номера варианта помноженному на 10 и округленному в большую сторону. Диапазон значений координат вводится пользователем при запуске программы. Для четных вариантов таблица формируется в Excel или другом оффлайновом аналоге. Для нечетных вариантов таблица формируется в таблицах google. Для заданных координат реализовать алгоритм метода наименьших квадратов (не используя готовые библиотеки для МНК) и построить гарфик (библиотека matplotlib).

Листинг (Решение):

С помошью множества выбирается два множества с уникальными элементами для x и y, далее по формуле, с помощью метода наименьших квадратов, высчитываются коэффициенты для линейной функции – “средней линии” для всех точек. Высчитав всё это, программа строит график.

```
import gspread
import random
import matplotlib.pyplot as plt
sa = gspread.service_account(filename="Token.json") # Подключение сервис-аккаунта
sheet = sa.open("Task_5")  # Открытие таблицы
worksheet = sheet.worksheet("xy")   # Работа с листом

values = list(map(int,input("Введите диапазон значений через пробел: ").split()))
```

Рисунок 18. Импортирование нужных для работы библиотек.

```
if len(values) == 2:
    minimum = min(values[0],values[1])
    maximum = max(values[1],values[0])
    if maximum - minimum >= 10:
        x = set()
        y = set()
        x_list = []
        y_list = []
        sum_xy = 0
        sum_x = 0
        sum_y = 0
        sum_xx = 0
        a = 0
        b = 0
        while len(x) < 10:
            z = random.randint(minimum,maximum)
            if z not in x:
                x_list.append(z)
                x.add(z)
        while len(y) < 10:
            z = random.randint(minimum, maximum)
            if z not in y:
                y_list.append(z)
                y.add(z)
        for i in range(len(x_list)):
            sum_xy += x_list[i]*y_list[i]
            sum_x += x_list[i]
            sum_y += y_list[i]
            sum_xx += x_list[i]**2
            worksheet.update_cell(i+1,1,x_list[i])
            worksheet.update_cell(i+1,2,y_list[i])
```

Рисунок 19. Нахождение уникальных рандомных координат и занесение их в гугл таблицу.

```
        a = (10*sum_xy - sum_x*sum_y)/(10*sum_xx-sum_x**2)
        b = (sum_y-a*sum_x)/10
        mnk = [[minimum,maximum],[a*minimum+b,a*maximum+b]]
        ax = plt.gca()
        ax.set_facecolor('black')
        plt.plot(mnk[0],mnk[1],c='yellow')
        plt.xlabel("Координаты X")
        plt.ylabel("Координаты Y")
        plt.title("Картина 'Ночное небо', автор Малевич К.")
        plt.scatter(x_list,y_list,c="aqua")
        plt.show()
    else:
        print("Диапазон мал")
```

Рисунок 20. Построение графиков

Результат выполнения программы:

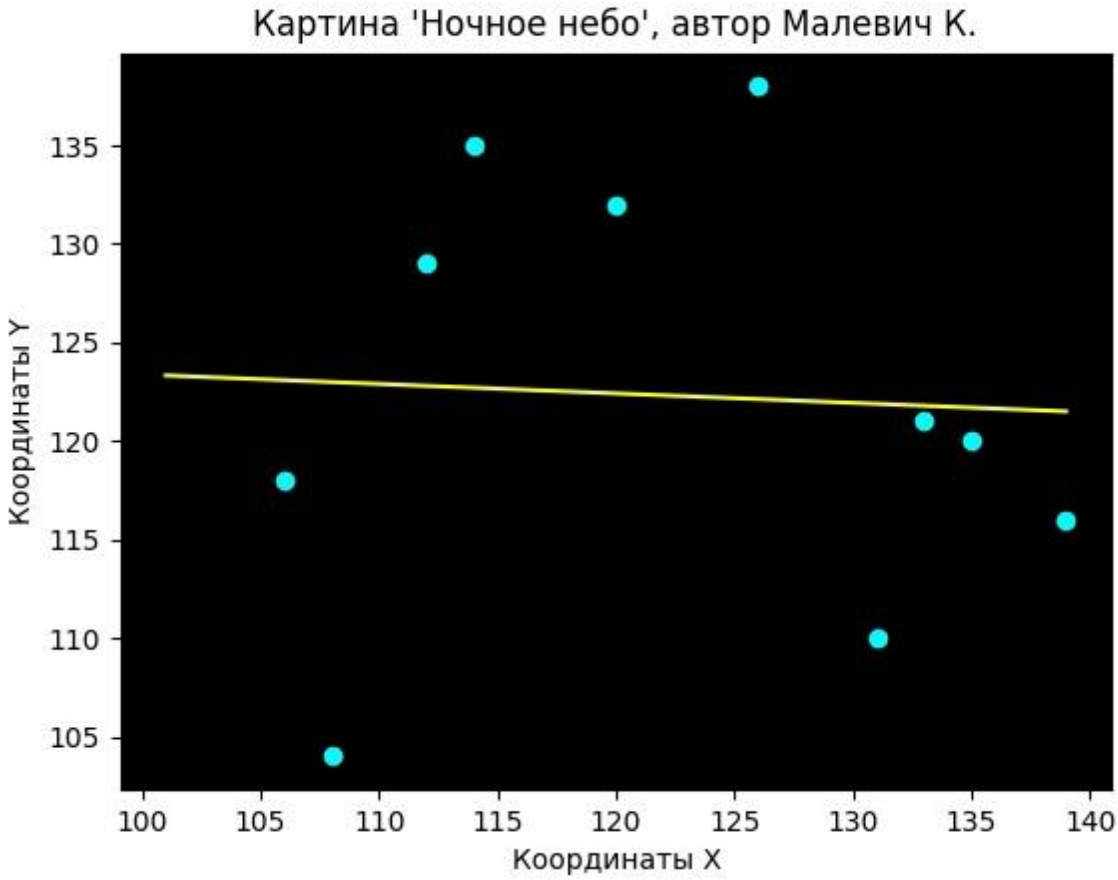

Рисунок 21. Результат выполнения программы. График.

	A	B
1	112	129
2	120	132
3	135	120
4	126	138
5	133	121
6	131	110
7	106	118
8	108	104
9	114	135
10	139	116

Рисунок 22. Результат выполнения программы. Таблица.

Вывод:
Для уникальности координат использовалась структура данных set(множество).
Для расчёта коэффициентов прямой используется метод наименьших квадратов.

Задание №6

Описание задания:

Без применения готовых библиотек, написать алгоритм, который позволяет вводить матрицу (указывается размер и значения элементов), а затем по желанию пользователя выполнять возведение в квадрат (если возможно),транспонировать (если возможно), для квадратной матрицы находить определитель.

Листинг (Решение):

Программа была реализована на основе математических определений действий над матрицами.

Транспонирование матрицы (Рисунок 23) – поворот матрицы, в программе он осуществлён посредством перестановки значений i-го столбца j-той строки и j-го столбца i-ой строки.

```
def transpon(array):
    transpon_array = []
    for i in range(columns):
        line = []
        for j in range(lines):
            line.append(0)
        transpon_array.append(line)
    for i in range(len(array)):
        for j in range(len(array[i])):
            transpon_array[j][i] = array[i][j]
    print("Транспонированная матрица :")
    for i in range(len(transpon_array)):
        for j in range(len(transpon_array[i])):
            print(transpon_array[i][j], end=" ")
        print()
```

Рисунок 23. Транспонирование матрицы.

Возведение матрицы в квадрат (Рисунок 24) - высчитывается в тройном цикле, в котором новое значениедля i j элемента матрицы высчитывается путём суммы произведений z-ых элементов i-го столбика с z-ых элементов j-ой строчки.

```
def square(array):
    if lines == columns:
        square_array = []
        for i in range(len(array)):
            line = []
            for j in range(len(array[i])):
                line.append(0)
            square_array.append(line)
        for i in range(len(array)):  # строка
            for j in range(len(array[i])):  # столбец
                for z in range(len(array)):  # номер числа внутри строки и столбца
                    square_array[i][j] += array[i][z] * array[z][j]
        print("Квадрат матрицы :")
        for i in range(len(square_array)):
            for j in range(len(square_array[i])):
                print(square_array[i][j], end=" ")
            print()
    else:
        print("Количество строк не совпадает с количеством столбцов")
```

Рисунок 24. Возведение матрицы в квадрат.

Функция нахождения детерминанты матрицы (Рисунок 25) рекурсивна, так как для того, чтобы найти детерминанту матрицы i-го порядка, нужно высчитать детерминанту миноров(матриц (i-1) порядка)

```
def det(array):
    if lines == columns:
        if len(array) == 2:
            return array[0][0] * array[1][1] - array[0][1] * array[1][0]
        else:
            sum = 0
            for i in range(len(array)):
                new_array = []
                for j in range(1, len(array)):
                    line = []
                    for k in range(len(array[j])):
                        if k != i:
                            line.append(array[j][k])
                    new_array.append(line)
                sum += array[0][i] * (-1) ** ((i + 1) + (0 + 1)) * det(new_array)
        return sum
    else:
        print("Определитель найти невозможно")
```

Рисунок 25. Функция нахождения детерминанты матрицы.

Результат выполнения программы:

```
Введите количество строк матрицы: 4
Введите количество столбцов матрицы: 4
Введите 1 строку через пробел: 1 2 3 4
Введите 2 строку через пробел: 5 6 7 8
Введите 3 строку через пробел: 9 10 11 12
Введите 4 строку через пробел: 13 14 15 16
Ваша матрица:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Введите команду, доступные команды: transpon, square, det, exit : transpon
```

```
Транспонированная матрица :
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
Введите команду, доступные команды: transpon, square, det, exit : square
Квадрат матрицы :
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

```
Введите команду, доступные команды: transpon, square, det, exit : det
Определитель матрицы : 0
Введите команду, доступные команды: transpon, square, det, exit : exit

Process finished with exit code 0
```

Рисунки 26,27,28 – результат выполнения программы.

Вывод:
Работа с матрицами без дополнительных библиотек не располагает к себе.