Поиск элемента в матрице+ сортировка

Теория


```
rows, cols = 3, 4

# rows - количество строк, cols - количество столбцов

matrix  = [[2, 3, 1, 0],
           [9, 4, 6, 8],
           [4, 7, 2, 7]]

for r in range(rows):
    for c in range(cols):
        print(matrix[r][c], end=' ')
    print()
```

1

Вывести на экран (в одну строку):

1.1.все элементы заданной строки

1.2.все элементы заданного столбца

1.3.все элементы главной диагонали

1.4.все элементы побочной диагонали

2

Заменить значения элементов:

2.1.четные на 0, нечетные на 1

2.2.кратные трем на 33

3

3. Определить:

3.1.сумму элементов заданной строки(столбца)

3.2.сумму элементов главной (побочной) диагонали

3.3.сумму всех элементов массива

4

Определить:

4.1.минимальный (максимальный) элемент заданной строки(столбца)

4.2.координаты минимального (максимального) элемента строки(столбца);

4.3.минимальный (максимальный) элемент главной (побочной)диагонали;

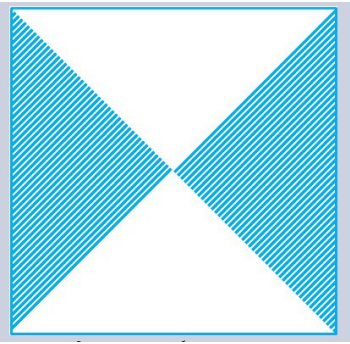
4.4.координаты минимального (максимального) элемента главной (побочной)диагонали;

4.5.минимальный (максимальный) элемент массива;

4.6.координаты минимального (максимального) элемента массива;

5

Напишите программу, которая выводит максимальный элемент в заштрихованной области квадратной матрицы.



6

Даны два двумерных целых массива одинаковых размеров.

создать третий массив такого же размера, каждый элемент которого равен сумме соответствующих элементов двух первых массивов.

7

В массиве 5x5 поменять местами элементы главной и побочной диагонали

8

Заполнить массив с помощью генератора:

0 1 2 3	0 5 10 15 20 25
4 5 6 7	1 6 11 16 21 26
8 9 10 11	2 7 12 17 22 27
12 13 14 15	3 8 13 18 23 28
	4 9 14 19 24 29

Основы сортировки в Python

Функция sorted() — это универсальный метод сортировки. В качестве обязательного параметра она принимает любой итерируемый объект и возвращает отсортированный список, созданный из его элементов. Эта функция не меняет исходный объект, а создаёт новый.

Синтаксис выглядит так:

```
sorted(iterable, key=None, reverse=False)
```

Параметры функции:

- iterable — обязательный. В него передаётся итерируемый объект, который вы хотите отсортировать (список, кортеж, строка, множество, замороженное множество).
- key — необязательный. Функция одного аргумента, которая будет применена к каждому элементу (по умолчанию None).
- reverse — необязательный. По умолчанию sorted() сортирует объект по возрастанию — но если поставить reverse=True, можно расположить элементы в обратном порядке.

В результате сортировки появился новый список, а исходный не изменяется.

Основы сортировки в Python

Синтаксис .sort():

list.sort(key=None, reverse=False)

Вместо list нужно указать название списка, к которому применяется метод.

Параметры у метода .sort() необязательные. Они аналогичны параметрам sorted():

- key — определяет небольшую функцию, в качестве аргумента принимающую элемент списка. Для каждого элемента она создаёт ключ сравнения — значение, по которому будут сравниваться эти элементы (по умолчанию — None).
- reverse — булевый аргумент, принимающий значения True или False. Если установлено значение True, список сортируется в обратном порядке (по умолчанию — False).

В результате выполнения вернул None, а сам список lst изменился

Функции в Python реализуют алгоритм Tim Sort, основанный на сортировке слиянием и сортировке вставкой.

Пузырьковая сортировка

Этот простой алгоритм выполняет итерации по списку, сравнивая элементы попарно и меняя их местами, пока более крупные элементы не «всплывут» в начало списка, а более мелкие не останутся на «дне».

Если взять самый худший случай (изначально список отсортирован по убыванию), затраты времени будут равны $O(n^2)$, где n — количество элементов списка.

```
def bubble_sort(nums):
    # Устанавливаем swapped в True, чтобы цикл запустился хотя бы один раз
    swapped = True
    while swapped:
        swapped = False
        for i in range(len(nums) - 1):
            if nums[i] > nums[i + 1]:
                # Меняем элементы
                nums[i], nums[i + 1] = nums[i + 1], nums[i]
                # Устанавливаем swapped в True для следующей итерации
                swapped = True
# Проверяем, что оно работает
random_list_of_nums = [5, 2, 1, 8, 4]
bubble_sort(random_list_of_nums)
print(random_list_of_nums)
```

Сортировка выборкой

Этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Наименьший элемент удаляется из второго списка и добавляется в первый.

Затраты времени на сортировку выборкой в среднем составляют $O(n^2)$, где n — количество элементов списка.

```
def selection_sort(nums):
    # Значение i соответствует кол-ву отсортированных значений
    for i in range(len(nums)):
        # Исходно считаем наименьшим первый элемент
        lowest_value_index = i
        # Этот цикл перебирает несортированные элементы
        for j in range(i + 1, len(nums)):
            if nums[j] < nums[lowest_value_index]:
                lowest_value_index = j
        # Самый маленький элемент меняем с первым в списке
        nums[i], nums[lowest_value_index] = nums[lowest_value_index], nums[i]
# Проверяем, что оно работает
random_list_of_nums = [12, 8, 3, 20, 11]
selection_sort(random_list_of_nums)
print(random_list_of_nums)
```

Сортировка вставками

Как и сортировка выборкой, этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Алгоритм перебирает второй сегмент и вставляет текущий элемент в правильную позицию первого сегмента.

Время сортировки вставками в среднем равно $O(n^2)$, где n — количество элементов списка.

```
def insertion_sort(nums):
  # Сортировку начинаем со второго элемента, т.к. считается, что первый элемент уже отсортирован
  for i in range(1, len(nums)):
    item_to_insert = nums[i]
    # Сохраняем ссылку на индекс предыдущего элемента
    j = i - 1
    # Элементы отсортированного сегмента перемещаем вперёд, если они больше  элемента для вставки
    while j >= 0 and nums[j] > item_to_insert:
      nums[j + 1] = nums[j]
      j -= 1
    # Вставляем элемент
    nums[j + 1] = item_to_insert
# Проверяем, что оно работает
random_list_of_nums = [9, 1, 15, 28, 6]
insertion_sort(random_list_of_nums)
print(random_list_of_nums)
```

Время сортировки

Алгоритм	Структура данных	Временная сложность			Вспомогательные данные
		Лучшее	В среднем	В худшем	В худшем
Быстрая сортировка	Массив	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Сортировка слиянием	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Пирамидальная сортировка	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Пузырьковая сортировка	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка вставками	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка выбором	Массив	O(n^2)	O(n^2)	O(n^2)	O(1)
Блочная сортировка	Массив	O(n+k)	O(n+k)	O(n^2)	O(nk)
Поразрядная сортировка	Массив	O(nk)	O(nk)	O(nk)	O(n+k)

9

Отсортировать строки двухмерного массива чисел по возрастанию значений элементов в первом столбце.

10

Отсортировать столбцы двухмерного массива чисел по возрастанию значений элементов в первой строке.

11

Первый столбец массива содержит номер года, второй – номер месяца, третий – число (день). Отсортировать все строки массива в порядке возрастания значения даты.