A. Итератор по RLE. 2021

Ограничение времени	4 секунды
Ограничение памяти	64.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Написать модуль, содержащий реализацию класса `RleSequence` для кодирования длин серий (Run-length encoding). Класс должен включать/перегружать следующие методы:

- `__init__(self, input_sequence)` — конструктор класса. По входному вектору `input_sequence` строится два вектора одинаковой длины. Первый содержит числа, а второй — сколько раз их нужно повторить. Для реализации рекомендуется использовать функцию `encode_rle` из второго домашнего задания. Кроме этих двух векторов в классе запрещается хранить любые объекты, размер которых зависит от длины исходного вектора.
    - input_sequence — одномерный numpy.array

Экземпляры класса должны поддерживать протокол итераций, причём порядок элементов, выдаваемый в процессе итерирования по экземпляру `RleSequence`, должен совпадать с порядком элементов в исходном `input_sequence`, который подавался в конструктор класса. Экземпляры должны поддерживать простое индексирование (положительные и отрицательные целые индексы) и взятие срезов с положительным шагом (третий параметр). Экземпляры должны поддерживать проверку на вхождение (`in` и `not in`).

Пример правильно работающего класса:

```
>>> rle_seq = RleSequence(np.array([1, 1, 2, 2, 3, 4, 5]))
>>> rle_seq[4]
3
>>> rle_seq[1:5:2]
array([1, 2])
>>> rle_seq[1:-1:3]
array([1, 3])
>>> 5 in rle_seq
True
>>> list(rle_seq)
[1, 1, 2, 2, 3, 4, 5]
```

Замечание. Некоторые операторы и функции перегружаются автоматически при определении других. Тем не менее, часто собственная реализация может работать эффективнее, чем созданная по умолчанию.

B. Линеаризация. 2021

Ограничение времени	0.1 секунд
Ограничение памяти	5.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Напишите модуль, содержащий реализацию итератора `linearize`, который принимает на вход любой итерируемый объект и линеаризует его, то есть раскрывает в нем все вложенности.

Пример правильно работающего кода:

```
>>> list(linearize([
...     4, "mmp", [8, [15, 1], [[6]], [2, [3]]], range(4, 2, -1)
... ]))
[4, "m", "m", "p", 8, 15, 1, 6, 2, 3, 4, 3]
```

C. Генератор батчей. 2021

Ограничение времени	0.4 секунд
Ограничение памяти	32.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Напишите класс `BatchGenerator`, который принимает на вход список последовательностей, размер батча и параметр `shuffle` и возвращает генератор, разбивающий входные последовательности на батчи заданного размера, а также случайным образом перемешивает их если `shuffle=True`.

Конструктор класса должен иметь следующий вид:

```
def __init__(self, list_of_sequences, batch_size, shuffle=False):
    """
    :param list_of_sequences: Список списков или numpy.array одинаковой длины
    :param batch_size: Размер батчей, на которые нужно разбить входные последовательности.
        Батчи последнего элемента генератора могут быть короче чем batch_size
    :param shuffle: Флаг, позволяющий перемешивать порядок элементов в последовательностях
    """

    pass
```

Примеры правильно работающего кода:

```
>>> bg = BatchGenerator(
...     list_of_sequences=[
            [1, 2, 3, 5, 1, 'a'],
            [0, 0, 1, 1, 0, 1]
        ], batch_size=2, shuffle=False
... )
...
>>> for elem in bg:
...     print(elem)
[[1, 2], [0, 0]]
```

```
[[3, 5], [1, 1]]
```

```
[[1, 'a'], [0, 1]]
```

Замечание: в принципе, реализовывать именно через `__init__` необязательно. Можно реализовывать и как итератор, и как генератор.

D. Генератор контекста. 2021

Ограничение времени	0.1 секунд
Ограничение памяти	5.0 Мб
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Напишите класс `WordContextGenerator`, который принимает на вход список строк и размер окна и возвращает генератор, возвращающий пары слов встречающиеся в одном окне размера `k`.

Конструктор класса должен иметь следующий вид:

```
def __init__(self, words, k):
    """
    :param words: Список слов
    :param window_size: Размер окна
    """
    pass
```

Примеры правильно работающего кода:

```
>>> s = ['мама', 'очень', 'хорошо', 'мыла', 'красивую', 'раму']
...
>>> for elem in WordContextGenerator(s, k=2):
...     print(elem)
('мама', 'очень')
('мама', 'хорошо')
('очень', 'мама')
('очень', 'хорошо')
('очень', 'мыла')
('хорошо', 'мама')
('хорошо', 'очень')
('хорошо', 'мыла')
('хорошо', 'красивую')
('мыла', 'очень')
('мыла', 'хорошо')
('мыла', 'красивую')
('мыла', 'раму')
```

```
('красивую', 'хорошо')
('красивую', 'мыла')
('красивую', 'раму')
('раму', 'мыла')
('раму', 'красивую')
```

Замечание: в принципе, реализовывать именно через `__init__` необязательно. Можно реализовывать и как итератор, и как генератор.