ВМК

Задание 5. Генераторы и итераторы.

Практикум 317 группы 2018-2019, осенний семестр

Начало выполнения задания: 05 ноября 2018 года.
Срок сдачи: **11 ноября 2018 года, 23:59.**

Решение каждой задачи должно быть описано в модуле `task_<номер задачи>.py`.

В систему anytask необходимо сдать `zip` архив, содержащий решения задач, называющийся `contest4_<фамилия студента>_<имя студента>.zip`.

Задачи основной части дополнительно сдаются в систему «Яндекс-контест», где проверяются с помощью автоматических тестов. Ссылку на страницу соревнования можно найти на странице курса. За каждую из задач основной части можно получить либо 5 баллов (вердикт 'OK' контеста), либо 0. Задание считается сданным, если хотя бы одна из задач получила вердикт 'OK'.

1. Написать собственный класс `RleSequence`, реализующий кодирование 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(x)` 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
   >>> my_list = []
   >>> for elem in rle_seq:
   ...     my_list.append(elem)
   >>> my_list
   [1, 1, 2, 2, 3, 4, 5]
   ```

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

2. Написать итератор `linearize`, принимающий на вход любой итерируемый объект и линеаризующий его, т.е. раскрывающий все вложенности.

   Пример правильно работающего кода:

   ```
   >>> my_list = list()
   >>> for elem in linearize([4, 'mmp', [8, [15, 1], [[6]], [2, [3]]], range(4, 2, -1)]):
   ...     my_list.append(elem)
   >>> my_list
   [4, 'm', 'm', 'p', 8, 15, 1, 6, 2, 3, 4, 3]
   ```

3. 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`.

   Класс должен включать в себя:

   - `__init__(self, list_of_sequences, batch_size, shuffle=False)` — конструктор класса.
     - `list_of_sequences` — список списков или `numpy.array` одинаковой длины.
     - `batch_size` — размер батчей, на которые нужно разбить входные последовательности. Батчи последнего элемента генератора могут быть короче чем `batch_size`.
     - `shuffle` — флаг, позволяющий перемешивать порядок элементов в последовательностях.

   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.

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

   Класс должен включать в себя:

   - `__init__(self, words, window_size)` — конструктор класса.
     - `words` — список слов.
     - `window_size` — размер окна.

   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
мылы, очень
мыла, хорошо
мыла, красивую
мыла, раму
красивую, хорошо
красивую, мыла
красивую, рама
рама, мыла
рама, красивую
```

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