Python (BSU FAMCS Fall'18)

Семинар 3

Преподаватели: Дмитрий Косицин, Светлана Боярович

Общие замечания ко всем заданиям

Свой тестирующий код можно размещать под условием `if __name__ == '__main__'` внизу файла. Такой код выполнится только если запустить этот файл, но не импортировать его.

Просьба использовать те имена для функций и файлов, которые указаны в замечаниях к заданиям.

Для отправки заданий выберите в anytask нужную задачу и там к сообщению прикрепите свое решение (один или несколько .py файлов).

Для тестирования интерфейса ваших заданий выложен специальный скрипт. Просьба также не оставлять debug print'ы в ваших программах, которые нужны исключительно для вывода отладочной информации.

Задание 1. (1.5 балла). Пусть у вас есть класс, представляющий собой узел односвязного списка `Node` (см. ниже).

Тогда односвязный список будет представлять собой последовательность узлов `Node`, где поле *next* либо является объектом типа `Node`, либо равно `None` (конец списка).

Списки могут быть вложенными – в этом случае значение `value` будет иметь тип `Node`.

Реализуйте следующее:

- Добавьте в класс проверки типов в конструкторе (используйте `assert`);
- Добавьте также свойства (property), позволяющие взять или изменить значения `_value` и `_next`;
- Добавьте «магический» метод `__iter__`, позволяющий проитерироваться по списку;
- Напишите функцию `flatten_linked_list`, которая разворачивает список *inplace* (см. пример *r3* ниже), т.е. вставляет вложенные списки в исходный, делая один цельный список.

Пример

```
class Node(object):
    def __init__(self, value, next_=None):
        self._value = value
        self._next = next_


r1 = Node(1)  # 1 -> None - just one node

r2 = Node(7, Node(2, Node(9)))  # 7 -> 2 -> 9 -> None

# 3 -> (19 -> 25 -> None) -> 12 -> None
r3 = Node(3, Node(Node(19, Node(25)), Node(12)))
r3_flattenned = flatten_linked_list(r3)  # 3 -> 19 -> 25 -> 12 -> None
r3_expected_flattenned_collection = [3, 19, 25, 12]
assert r3_expected_flattenned_collection == list(r3_flattenned)
```

Задание 2. (2 балла). Реализуйте класс `MidSkipQueue`, представляющий собой очередь, в которой хранятся только первые k и последние k добавленных элементов. В очереди должно быть реализовано следующее:

- конструктор, принимающий первым аргументом параметр k (проверьте, что он имеет допустимое значение), а вторым *опциональным* аргументом – *iterable* элементов, на основе которых нужно построить очередь. В качестве *iterable* может быть как некоторый список или кортеж, так и генератор;

- метод, преобразующий очередь в строку: переопределите «магический» метод `__str__`, ограничьте вывод при больших k (используйте модуль `pprint` при необходимости);
- метод, возвращающий копию объекта;
- операторы, позволяющие сравнивать на равенство и неравенство 2 очереди;
- «магический» метод `__iter__`, позволяющий проитерироваться по объекту;
- «магический» метод, возвращающий длину очереди;
- «магический» метод, позволяющий обратиться к элементу по индексу, в том числе отрицательному (используйте `assert` для проверки корректности индекса), а также взять слайс элементов;
- метод `index`, возвращающий индекс элемента в очереди или -1, если такого элемента нет;
- «магический» метод, позволяющий проверить, содержится ли некоторый элемент в очереди;
- метод `append`, который в качестве агрумента принимает один и более (переменное число) объектов (не *iterable*!) и добавляет все элементы в очередь;
- оператор сложения с *iterable* элементов.

Обратите внимание, что удаление из очереди реализовывать не требуется.
Постарайтесь переиспользовать как можно больше уже имеющихся методов.
Пожелание к амортизированному времени добавления элемента – $O(1)$.
Класс сохраните в файле *mid_skip_queue.py*.

Пример

```
q = MidSkipQueue(1)
q.append(-1)  # q: [-1]
q += (-2, -3)  # q: [-1, -3] - the first and the last remain
q.append(4)  # q: [-1, 4] - the last item has been replaced
```

Задание 3. (1 балл). От класса `MidSkipQueue` унаследуйте класс `MidSkipPriorityQueue`, в котором при добавлении элементов в очередь будет учитываться их значение так, что будут храниться не более k наименьших и k наибольших добавленных элементов. В начале очереди храните наименьший элемент. Предполагайте, что добавляемые элементы реализуют все необходимые операторы сравнения.

Постарайтесь переиспользовать как можно больше уже имеющихся методов.
Пожелание к амортизированному времени добавления элемента – $O(k)$.
Класс сохраните в файле *mid_skip_queue.py*.

Пример

```
q = MidSkipPriorityQueue(1)
q.append(-1)  # q: [-1]
q += (-2, -3)  # q: [-3, -1] - the smallest and the largest items
q.append(4)  # q: [-3, 4] - the largest item is replaced
q.append(-5)  # q: [-5, 4] - the smallest item is replaced
```

Задание 4. (1.5 балла). Напишите unit-тесты для каждого из классов – `MidSkipQueue` и `MidSkipPriorityQueue`. Используйте библиотеки `unittest` или `pytest`.