Домашнее задание №5

Дедлайн 1 (12 баллов): 11 октября, 13:00
Дедлайн 2 (6 баллов): 18 октября, 13:00

Домашнее задание нужно написать на Python 3 и сдать в виде одного файла. Правило именования файла: `name_surname_05.py`. Например, если вас зовут Иван Петров, то имя файла должно быть: `ivan_petrov_05.py`.

Обратите внимание, что при выполнении задания можно использовать только то, что мы уже успели обсудить в рамках курса.

1 Следующие задания нужно реализовать с помощью коллекций из модуля `collections`.

а Объявите именованный кортеж `Factor` с двумя полями `elements` и `levels`. `Factor` описывает список, закодированный числовой последовательностью. Реализуйте функцию `factor`, которая принимает список элементов и возвращает его в виде кортежа типа `Factor`.

```
>>> factor(["a", "a", "b"])
Factor(elements=[0, 0, 1], levels=OrderedDict([('a', 0), ('b', 1)]))
>>> factor(["a", "b", "c", "b", "a"])
Factor(elements=[0, 1, 2, 1, 0], levels=OrderedDict([('a', 0), ('b', 1), ('c', 2)]))
```

Номера уровней в возвращаемом кортеже должны соответствовать порядку уникальных элементов во входном списке.

б Декоратор `functools.lru_cache` позволяет запомнить фиксированное количество результатов последних вызовов функции. Префикс LRU (*англ.* least recently used) означает, что в случае переполнения кеша вытесняется элемент, обращение к которому на чтение или запись максимально отдалено от текущего обращения. Такой элемент всегда единственный.

Реализуйте декоратор `lru_cache`. Пример работы декоратора:

```
>>> @lru_cache
... def fib(n):
...     return n if n <= 1 else fib(n - 1) + fib(n - 2)
...
>>> fib(10)
55
```

Декоратор должен принимать один ключевой аргумент `maxsize` — максимальное количество элементов кеше, по умолчанию равный `64`.

Кроме того, к функции обёртке следует добавить два метода:

- Метод `cache_info` возвращает кортеж `CacheInfo` из четырёх элементов:

```
#                              максимальный размер
>>> fib.cache_info()  #         v
CacheInfo(hits=8, misses=11, maxsize=64, currsize=11)
#         ^            ^                    ^
#         |            |                текущий размер
#    попали в кеш   пересчитали значение
```

- Метод `cache_clear` очищает кеш и сбрасывает счётчики:

```
>>> fib.cache_clear()
>>> fib.cache_info()
CacheInfo(hits=0, misses=0, maxsize=64, currsize=0)
```

Для реализации вам потребуется несколько методов класса `OrderedDict`[1] , которые мы не обсуждали на лекции.

- Метод `OrderedDict.popitem` удаляет из словаря первый или последний добавленный элемент:

```
>>> cache = OrderedDict([("foo", 3), ("bar", 3)])
>>> cache.popitem(last=False)  # удалить первый элемент
('foo', 3)
>>> cache["boo"] = 3
```

- Метод `OrderedDict.move_to_end` перемещает указанный ключ в начало или в конец списка ключей:

```
>>> cache
OrderedDict([('bar', 3), ('boo', 3)])
>>> cache.move_to_end("boo", last=False)  #  переместить в начало
>>> cache
OrderedDict([('boo', 3), ('bar', 3)])
```

в Реализуйте функцию `group_by`, группирующую переданную ей последовательность объектов по функции-ключу. Результатом функции `group_by` является словарь, в котором по ключу `k` содержится список объектов, на которых функция-ключ вернула значение `k`.

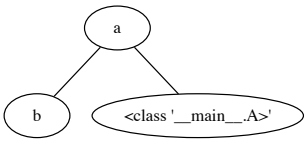
```
>>> dict(group_by(["foo", "boo", "barbra"], len))
{3: ['foo', 'boo'], 6: ['barbra']}
```

г Напишите функцию `invert`. Функция принимает словарь и возвращает новый словарь, в котором для каждого значения из исходного словаря содержится множество соответствующих ему ключей.

```
>>> dict(invert({"a": 42, "b": 42, "c": 24}))
{24: {'c'}, 42: {'b', 'a'}}
```

[1] `https://docs.python.org/3.4/library/collections.html#collections.OrderedDict`

2 Неориентированный граф в Python удобно представлять в виде словаря, ключи которого — вершины графа, а значения — соответствующие списки смежности. Часто вершины графа как-то занумерованы и представляются в виде чисел. Также у вершин есть текстовые метки, которые хранятся в отдельном списке. Например:




```
>>> labels = ["a", "b", "c"]
>>> g = {0: [1, 2], 1: [0], 2: [0]}
```

Напишите функцию `export_graph`, которая принимает граф, список меток для каждой вершины графа и путь к файлу и записывает граф в файл в формате DOT. Формат DOT по сути представляет из себя список ребер с метаданными. Для графа выше результат функции `export_graph` должен выглядеть как:

```
graph {
0 [label="a"]
0 -- 1
0 -- 2
1 [label="b"]
2 [label="c"]
}
```

Порядок следования строчек внутри оператора `graph` не важен.

Обратите внимание, что для каждой пары вершин из графа в DOT файле должно остаться не более одного ребра. Направление ребра в файле значения не имеет, то есть ребро `(0, 1)` можно записать либо как `0 -- 1`, либо как `1 -- 0`.

3 Опечатки — ежедневная проблема всех пользователей компьютера. Борьбе с опечатками посвящено большое количество публикаций[2]. Мы попробуем решить задачу исправления опечаток для списка слов.

а Напишите функцию `build_graph`, которая по заданному списку слов `words` и значению `mismatch_percent` строит неориентированный граф, называемый графом Хемминга. Каждое слово из списка `words` это метка, соответствующей ему вершины в графе. Ребро между двумя вершинами проводится, если:

- длины строк, соответствующих вершинам, равны,
- `d <= mismatch_percent * n / 100`, где `d` — расстояние Хемминга между двумя строками, а `n` — их длина.

```
>>> words = ["hello", "helol", "ehllo", "tiger", "field"]
>>> g = build_graph(words, mismatch_percent=50.)
>>> g
{0: [1, 2], 1: [0], 2: [0], 3: [], 4: []}
```

Обратите внимание, что одинаковым словам в списке `words` соответствуют **разные** вершины в итоговом графе.

б Напишите функцию `find_connected_components`, которая по заданному неориентированному графу строит список его компонент связности:

```
>>> find_connected_components(g)
[[0, 1, 2], [3], [4]]
```

в Напишите функцию `find_consensus`, которая принимает на вход список слов одинаковой длины. Она возвращает строку, в которой i-й символ — наиболее часто встречающаяся буква, стоящая в i-й позиции, во всех словах из списка. Найденная строка называется *консенсусной*. Если консенсусных строк несколько, можно вернуть любую.

```
>>> find_consensus(["hello", "helol", "ehllo"])
'hello'
>>> find_consensus(["bug", "bow", "bag", "bar"])
'bag'
```

г Напишите функцию `correct_typos`, которая принимает на вход список слов и значение `mismatch_percent` и возвращает новый список, где слова с опечатками исправлены.

Исправление слов происходит по следующему алгоритму.

1. Строим по списку слов граф Хемминга, используя переданное значение `mismatch_percent`.
2. Ищем в полученном графе компоненты связности.
3. Для каждой найденной компоненты связности строим консенсусную строку и исправляем с помощью неё все слова из этой компоненты. Исправить в данном случае означает заменить слово с опечаткой на соответствующую консенсусную строку.

```
>>> words = ["hello", "helol", "ehllo", "tiger", "field", "abracadabra"]
>>> correct_typos(words, mismatch_percent=50.)
['hello', 'hello', 'hello', 'tiger', 'field', 'abracadabra']
```

[2] http://scholar.google.com/scholar?q=spelling+correction