Генератор текста

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Ваша задача состоит в том, чтобы написать автоматический генератор текстов с помощью цепей Маркова (http://ru.wikipedia.org/wiki/Цепь_Маркова).
Примерный алгоритм работы программы заключается в следующем. На стадии обучения программа должна для каждого слова w_1 посчитать, с какой частотой произвольное другое слово w_2 встречается после слова w_1. Далее, на стадии генерации текста, программа должна после очередного слова генерировать следующее в соответствии с посчитанным распределением частот. Аналогично можно генерировать очередное слово в зависимости от двух предыдущих и так далее.

Предложения и знаки препинания должны быть максимально похожи на нормальные тексты: предложение начинается с заглавный буквы, пробел стоит только после знака препинания, а не до него, не встречается несколько знаков препинания подряд и т.п.

Детали реализации

Первый этап работы программы — токенизация текста, то есть разбиение его на составные части (токены), которые далее будут рассматриваться как «слова». Токеном будем называть последовательности символов, целиком состоящие из букв, последовательности символов, целиком состоящие из цифр, и отдельные символы. При разбиении текста на токены нужно брать максимально длинные токены из возможных (то есть "abc" нужно рассматривать как один токен, а не как "a", "b", "c").

Далее, программа должна уметь подсчитывать частоты слов после заданной цепочки слов. При этом у программы есть параметр D — максимальная глубина, — и программа должна подсчитывать частоты для всех последовательностей не длинее D. Это нужно для того, чтобы при генерации была возможность стабильно генерировать начала предложений, когда сгенерированная история еще короткая. Статистику нужно подсчитать в том числе и для пустой истории (это по сути просто частоты слов).

Для программы также нужно написать юнит-тесты (в том же файле). Юнит-тесты должны проверять работу всех содержательных частей программы.

Формат ввода

У программы должно быть 4 режима работы: токенизация текста, подсчет вероятностей, генерация текста и юнит-тестирование. Режим работы и параметры работы задаются в первой строке входного файла. Все остальные данные находятся в последующих строках. Весь текст дается в

кодировке UTF-8.
В режиме токенизации программа получает на вход одну строку текста и разбивает ее на токены. В этом режиме в первой строке написано слово "`tokenize`". На выход нужно вывести токены, разделенные переносами строки (так как на входе только одна строка, переносов в ней и среди токенов нет).

В режиме подсчета вероятностей программа получает на вход несколько строк и считает вероятности цепочек. В этом режиме в первой строке написано слово "`probabilities`", а также аргумент "`--depth`", задающий максимальную глубину цепочек. Программа должна подсчитать вероятности и вывести их на выходной поток. Вывод нужно отформатировать следующим образом. Весь вывод должен быть разбит на блоки, каждый блок соответствует цепочке истории; блоки должны быть отсортированы лексикографически. В каждом блоке сначала идет цепочка истории (токены в цепочке, разделенные пробелами), а потом на отдельных строках — вероятности слов после этой цепочки. Для каждого слова нужно сделать отступ в 2 пробела, потом вывести слово, вывести ": ", потом частоту с ровно 2-мя знаками после запятой. При подсчете вероятностей нужно учитывать только токены, состоящие из букв, регистр менять не нужно. Кроме того, нужно сделать, чтобы история у разных строк была независимой (то есть при переходе к следующей строке история сбрасывается). Слова, для которых вероятность равна 0, выводить не нужно. Обратите внимание, что первым всегда идет блок, соответствующий глубине 0 и он обозначается пустой строке (в примере эта пустая строка не отображается по техническим причинам).

В режиме генерации текста программа получает на вход несколько строк текста, обучается на них и генерирует текст заданной длины. В этом режиме в первой строке написано слово "`generate`", а также аргумент "`--depth`", задающий максимальную глубину цепочек и аргумент "`--size`", задающий примерное количество слов для генерации. На выход нужно вывести сгенерированный текст.

В режиме юнит-тестирования программа запускает внутренние тесты. В этом режиме в первой строке написано слово "`test`".

Пример 1

Ввод	Вывод

`tokenize`
`Hello, world!`
	`Hello`
	`,`
	
	`world`
	`!`

Пример 2

Ввод	Вывод

`probabilities --depth 1`
`First test sentence`
`Second test line`
	`  First: 0.17`
	`  Second: 0.17`
	`  line: 0.17`

```
  sentence: 0.17
  test: 0.33
First
  test: 1.00
Second
  test: 1.00
test
  line: 0.50
  sentence: 0.50
```

Примечания

При решении нельзя пользоваться библиотекой `tokenize`.
Для разбора аргументов удобно пользоваться библиотекой `argparse`. Самостоятельно изучите, как устроены под-парсеры и как их использовать для создания разных режимов работы программы.

При выводе частот удобно воспользоваться методом format у строки.

В режимах `generate` и `test` фактически проверяется лишь то, что программа успешно завершилась. Но это не означает, что программе не обязательно уметь генерировать тексты или к ней не обязательно писать тесты.