**Практика «Генерация текстов»**

Скачайте проект TextAnalysis. Следующие три задачи нужно будет выполнять в этом одном проекте.

Конечная цель — сделать алгоритм продолжения текста, который, обучившись на большом тексте, будет способен по первому введенному слову предложить правдоподобное продолжение фразы.

Вся задача разбита на 3 этапа:

1. Разбиение анализируемого текста на предложения и слова.
2. Составление N-граммной модели текста по списку предложений.
3. Генерация текста по N-граммной модели.

Например, в результате анализа текста о Гарри Поттере наш несложный алгоритм должен быть способен генерировать новые, не содержащиеся в исходном тексте, но при этом довольно реалистичные фразы:

harry and ron were delighted to hear all about the sorcerer's stone

boy who kept losing his toad sniffed once or twice a minute all summer

**Тестирование и отладка**

В этой задаче много тонких нюансов реализации, в которых легко ошибиться. Поэтому в проекте есть специальная система тестов, проверяющих корректность реализации каждого этапа. Тесты находятся в классах, с именами заканчивающимися на \_Tests.cs.

Все тесты запускаются при запуске приложения благодаря этой строчке в классе Program.cs:

Вместо многоточия можно передать разные опции, в частности можно указать набор тестов, которые нужно запускать. Вы можете указать там только тесты на ту задачу, которой занимаетесь в текущий момент.

Также вы можете поставить точку останова прямо в код неработающего теста и отладить ваш алгоритм по шагам.

При отправке вашего финального решения на проверку, система проверит работоспособность сначала этих тестах, а затем проверит, что ваш алгоритм работает правильно и на большом тексте про Гарри Поттера.

=========================================================

## Практика «Парсер предложений»

В этом задании нужно реализовать метод в классе SentencesParserTask. Метод должен делать следующее:

1. Разделять текст на предложения, а предложения на слова.

a. Считайте, что слова состоят только из букв (используйте метод char.IsLetter) или символа апострофа ' и отделены друг от друга любыми другими символами.

b. Предложения состоят из слов и отделены друг от друга одним из следующих символов .!?;:()

1. Приводить символы каждого слова в нижний регистр.
2. Пропускать предложения, в которых не оказалось слов.

Метод должен возвращать список предложений, где каждое предложение — это список из одного или более слов в нижнем регистре.

**Практика «Частотность N-грамм»**

Продолжайте работу в **том же проекте.**

N-грамма — это N соседних слов в одном предложении. 2-граммы называют биграммами. 3-граммы — триграммами.

Например, из текста: "She stood up. Then she left." можно выделить следующие биграммы "she stood", "stood up", "then she" и "she left", но не "up then". И две триграммы "she stood up" и "then she left", но не "stood up then".

По списку предложений, составленному в прошлой задаче, составьте словарь самых частотных продолжений биграмм и триграмм. Это словарь, ключами которого являются все возможные начала биграмм и триграмм, а значениями — их самые частотные продолжения.

Более формально так:

Для каждой пары (key, value) из словаря должно выполняться одно из следующих условий:

1. В тексте есть хотя бы одна биграмма (key, value), и для любой другой присутствующей в тексте биграммы (key, otherValue), начинающейся с того же слова, value должен быть лексикографически меньше otherValue.
2. Либо в тексте есть хотя бы одна триграмма (w1, w2, value), такая что w1 + " " + w2 == key и для любой другой присутствующей в тексте триграммы (w1, w2, otherValue), начинающейся с той же пары слов, value должен быть лексикографически меньше otherValue.

Для лексикографического сравнения используйте встроенный в .NET способ сравнения Ordinal, например с помощью метода string.CompareOrdinal.

Такой словарь назовём N-граммной моделью текста.

Реализуйте этот алгоритм в классе FrequencyAnalysisTask.

Все вопросы и детали уточняйте с помощью примера ниже и тестов.

**Пример**

По тексту a b c d. b c d. e b c a d. должен быть составлен такой словарь:

"a": "b"

"b": "c"

"c": "d"

"e": "b"

"a b": "c"

"b c": "d"

"e b": "c"

"c a": "d"

Обратите внимание:

* из двух биграмм "a b" и "a d", встречающихся однократно, в словаре есть только пара "a": "b", как лексикографически меньшая.
* из двух встречающихся в тексте биграмм "c d" и "c a" в словаре есть только более частотная пара "c": "d".
* из двух триграмм "b c d" и "b c a" в словаре есть только более частотная "b c": "d".

======================================================

**Практика «Продолжение текста»**

Продолжайте работу в том же проекте.

В классе TextGeneratorTask реализуйте алгоритм продолжения текста по N-граммной модели.

Описание алгоритма:

На вход алгоритму передается словарь nextWords, полученный в предыдущей задаче, одно или несколько первых слов фразы phraseBeginning и wordsCount — количество слов, которые нужно дописать к phraseBeginning.

Словарь nextWords в качестве ключей содержит либо отдельные слова, либо пары слов, соединённые через пробел. По ключу key содержится слово, которым нужно продолжать фразы, заканчивающиеся на key.

Алгоритм должен работать следующим образом:

1. Итоговая фраза должна начинаться с phraseBeginning.
2. Алгоритм дописывает по одному слову к фразе wordsCount слов следующим образом:

a. Если фраза содержит как минимум два слова и в словаре есть ключ, состоящий из двух последних слов фразы, то продолжать нужно словом, хранящемся в словаре по этому ключу.

b. Иначе, если в словаре есть ключ, состоящий из одного последнего слова фразы, то продолжать нужно словом, хранящемся в словаре по этому ключу.

c. Иначе, нужно закончить генерирование фразы.

Проверяющая система сначала запустит эталонный способ разделения исходного текста на предложения и слова, потом эталонный способ построения словаря наиболее частотных продолжений из предыдущей задачи, а затем вызовет реализованный вами метод. В случае ошибки вы увидите исходный текст, на котором запускался процесс тестирования.

Если запустить проект на выполнение, он предложит ввести начало фразы и сгенерирует продолжение. Позапускайте алгоритм на разных текстах и разных фразах. Результат может быть интересным!

**О применении N-граммных моделей**

Подобные N-граммные модели текстов часто используются в самых разных задачах обработки текстов. Когда поисковая строка предлагает вам продолжение вашей фразы — скорее всего это результат работы подобного алгоритма.

Сравнивая частоты N-грамм можно сравнивать тексты на похожесть и искать плагиат.

Опираясь на N-граммные модели текстов можно улучшать алгоритмы исправления опечаток или автокоррекции вводимого текста.