Задачки для code review.

Программу на Fortran77 можно написать на любом языке.
Но мы не будем этого делать.

1. Общие принципы

-- Код задачи должен быть рабочим, это необходимое условие, но не достаточное.
-- Код лучше писать в соответствии с требованиям pep8 (https://www.python.org/dev/peps/pep-0008/). Любая IDE для python поможет с этим.

-- Процесс сдачи:
- Создать пустой приватный репозиторий на github.com
- Добавить туда семинариста (ivanaxe)
- Сделать начальный коммит в ветку master (пустой, или файлик с ФИО)
- Создать ветку dev от master и вести дальнейшую разработку в ней
- Написать и запушить код в ветке dev
- Создать pull request из dev в master и не вливать его. В нем будет проходить обсуждение задачи.
- While (семинарист не влил ревью):
- Дождаться комментов
- Исправить код, сделать коммит(-ы) в dev, запушить.
- Получить зачет по review.

2. Задача 1 “криптография для самых маленьких”

Нужно написать программу, которая шифрует/дешифрует тексты на латинице различными алфавитными шифрами замены.

Предполагается, что такие алгоритмы шифрования безразличны к небуквенным символам (пробелы, знаки препинания), а также к регистру (большие/маленькие буквы). Допустимо или сразу приводить все строки к такому виду (один регистр, только буквенные символы) или оставлять регистр/небуквенные символы как есть (с точки зрения криптографии это порочная практика)

Шифрование

./encryptor.py encode --cipher {caesar|vigenere} --key {<number>|<word>} [--input-file input.txt] [--output-file output.txt]

Зашифровать входное сообщение.

Аргументы:

--cipher - тип шифра: caesar (Шифр Цезаря) или vigenere (Шифр Виженера).

--key - ключ шифра. Для шифра Цезаря - число, соответствующее сдвигу, для шифра Виженера - слово, которое задает сдвиги (*для определенности будем считать, что оно написано строчными буквами*).

--input-file (*необязательный аргумент*) - путь ко входному файлу с текстом. Если не указан, текст вводится с клавиатуры.

--output-file (*необязательный аргумент*) - путь к выходному файлу (*файла может не быть, и его тогда надо создать. Python с этим справляется сам при открытии файла на запись*). Если не указан, текст выводится в консоль.

Дешифрование

./encryptor.py decode --cipher {caesar|vigenere} --key {<number>|<word>} [--input-file input.txt] [--output-file output.txt]

Расшифровать входное сообщение, зная шифр и ключ, с которым оно было зашифровано.

Аргументы:

--cipher - тип шифра: caesar или vigenere.

--key - ключ шифра.

--input-file (*необязательный аргумент*) - путь ко входному файлу с текстом. Если не указан, текст вводится с клавиатуры.

--output-file (*необязательный аргумент*) - путь к выходному файлу. Если не указан, расшифрованное сообщение выводится в консоль.

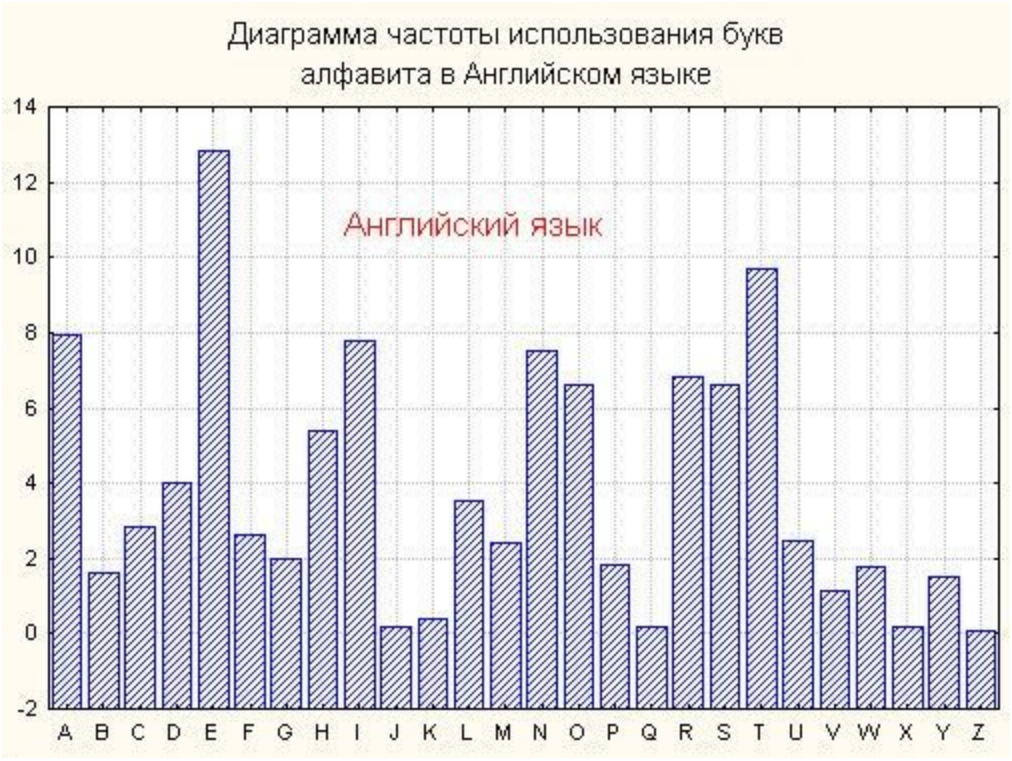
Взлом

Безусловно, дешифровать сообщение с известным алгоритмом и ключом шифрования - это полезно. Но давайте попробуем расшифровать сообещение, зашифрованное известным алгоритмом без знания ключа (атака на ключ)

В обязательной части предлагается расшифровать шифр Цезаря без знания сдвига. Если бы мы расшифровывали сообщение “руками”, то мы бы могли просто выписать все варианты дешифрования (благо, вариантов сдвига не так много - всего 26), посмотреть глазами, и выбрать, какой из них выглядит как настоящий текст. Компьютер же не знает, что такое “как настоящий текст”, поэтому мы его обучим - построим простейшую языковую модель.

Примером простейшей модели может быть частотность символов.

Идея. Скормим программе какой-нибудь достаточно большой текст (например, сонеты Шекспира) и посмотрим на распределение различных букв (такой график еще называется *гистограммой*).



Видно, что какие-то буквы (A, E, T…) встречаются намного чаще остальных.

Теперь вернемся к нашей попытке подобрать параметр сдвига. Для всех вариантов сдвига посчитаем частоты символов в варианте расшифровки для этого сдвига. Какие-то гистограммы больше похожи на “правильную”, какие-то меньше. Та, которая больше всего похожа на “правильную”, скорее всего и будет соответствовать искомому сдвигу. Осталось формализовать понятие “похожести” гистограмм и все хорошо.

Допустим, у нас есть два набора частот букв: $A = \{a_i\}_{i=1}^{N}$ и $B = \{b_i\}_{i=1}^{N}$. Так как это частоты, нормированные на длины строк, то $\sum_{i=1}^{N} a_i = \sum_{i=1}^{N} b_i = 1$.

Какие есть варианты меры схожести:

- $F(A, B) = \sum_{i=1}^{N} |a_i - b_i|$
- $F(A, B) = \sum_{i=1}^{N} (a_i - b_i)^2$
- $F(A, B) = \sum_{i=1}^{N} (a_i - b_i)^p$, p - положительное вещественное.
- Можно пофантазировать

Команды для обучения и взлома имеют следующий вид:

./encryptor.py train --text-file {input.txt} --model-file {model}

Проанализировать текст и построить языковую модель

Аргументы:

--text-file (*необязательный аргумент*) - путь ко входному файлу с текстом. Если не указан, текст вводится с клавиатуры.

--model-file - путь к файлу модели, куда будет записана вся та статистика, которую вы собрали по тексту (*файл может не существовать, тогда его необходимо создать на лету*). Содержимое этого файла можно формировать руками, можно использовать представление **json** (и одноименный модуль питона), либо **yaml** или модуль **pickle**

./encryptor.py hack [--input-file input.txt] [--output-file output.txt] --model-file {model}

Попытаться расшифровать текст.

Аргументы:

--input-file (*необязательный аргумент*) - путь ко входному файлу с текстом. Если не указан, текст вводится с клавиатуры.

--output-file (*необязательный аргумент*) - путь к выходному файлу (*файл может не существовать, тогда его необходимо создать на лету*). Если не указан, расшифрованное сообщение выводится в консоль.

--model-file - путь к файлу модели, которая будет использоваться.

2. Возможные бонусы

- Добавить в шифратор/дешифратор шифр Вернама
- Научиться взламывать шифр Виженера с помощью индексов совпадений.
- Все вышеописанные идеи взлома хорошо работают, если зашифрованный текст достаточно большой. Для коротких текстов можно применять следующие идеи:
    - *Словари.* Если запомнить из большого текста помимо частот буквы еще и слова, то тогда можно просто выбрать правильный сдвиг с помощью поиска по словарю. При этом нужно уметь ограничить размер модели только словами большой частотности (либо превышающими какой-то порог по частоте, либо топ-N. Ограничения нужно сделать дополнительными аргументами программы). Нужно не забывать, что некоторые последовательности букв можно раскодировать в несколько слов сразу (CF -> OR или BE).
    - *N-граммы.* Помимо частот отдельных символов будем считать последовательности из N символов подряд (N-граммы). Допустим, N = 3. Тогда для каждой пары из N-1 = 2 букв можно посчитать частоты разных вариантов для третьей (например, самая частотная для пары TH буква будет E, а для пары EQ - U). Тогда при переборе сдвигов можно

посчитать такую величину (в статистике она называлась бы *правдоподобием*):

$L(\{w_i\}) = \sum_{i=2}^{N} freq(w_i \mid w_{i-2}, w_{i-1})$, где $freq(x \mid y, z)$ - частота буквы x, при фиксированной впереди стоящей паре букв y,z. Максимум правдоподобия соответствует “правильному” сдвигу.

- Добавить в шифратор/дешифратор/взлом поддержку кириллицы, пробелов и знаков препинания, чтобы они каким-то образом циклически друг в друга переходили.