Документация к программе «Шифратор»

Используемый ЯП: Python (версия 3.7).

Среда разработки: Pycharm.

Тип: консольное приложение (графический интерфейс отсутствует).

Основные функции:

1) шифрование файлов;

2) дешифрование файлов;

3) взлом шифра и расшифровка текста.

Импортированные модули:

- random

Общепрграммное. Происходящее в коде вообще. В программе реализованы три различных алгоритма шифрования и дешифрования: шифрования Цезаря, Вижинера, Вернама, а также один алгоритм взлома — метод частотного анализа для русского языка.

В начале работы пользователь выбирает режим работы, вводя соответствующую цифру: 1 — режим шифрования, 2 — дешифрования, 3 — взлома (здесь и далее при каждом запросе ввода программа подсказывает пользователю, какие варианты ответа у него есть и что он может ввести; в случае неверного ввода корректное поведение программы не гарантируется (UB)). При этом, без участия пользователя заполняются словари, хранящие в себе алфавит: alphabet_liter_number – словарь вида (буква; номер), alphabet_number_liter – словарь вида (номер; буква), frequencies – словарь вида (буква; относительная частота использования буквы в языке). По умолчанию, словари заполняются для алфавита байтового кода, т. е. чисел из отрезка [0; 255] (номера присваиваются в 1-индексации), при этом словарь frequencies остаётся default. Далее, действия программы зависят от выбранного режима работы.

1-2. Пользователю предлагается выбрать алгоритм шифрования (дешифрования): 1- Цезаря, 2- Вижинера, 3- Вернама. Затем программа спрашивает у пользователя путь к файлу, и считывает его в виде строки path. Строка передаётся в функцию reading(path), которая считывает байтовый код файла по пути path и возвращает его в виде массива байтов bytearray. В зависимости от конкретного алгоритма, пользователь также вводит либо сдвиг (шифр Цезаря), либо ключевое слово (последовательность целых чисел из отрезка [0; 255]) (шифр Вижинера и дешифровка шифра Вернама), либо не вводит ничего. Пользователю предлагается возможность вывести байтовый код считанного файла в консоль (функция print_or_not_print()). Затем, происходит обработка файла в соответствии с выбранным алгоритмом (подробное описание — см. далее). Далее, пользователю предлагается вывести результат работы (т. е. изменённую последовательность байтов) в консоль (print_or_not_print). Далее, у пользователя спрашивается, что делать с полученной последовательностью: 0 — ничего, 1 — перезаписать переданный файл, 2 — считать путь к новому файлу, и записать зашифрованные данные в него вместо имеющегося там содержимого (функция writing()). Затем в консоль выводится мини-отчёт: количество символов в байтовом коде переданного файла и ключ/ключевое слово.

3. По умолчанию, происходит перезапись алфавита: теперь алфавитом становится «расширенный» русский алфавит, т. е. русский алфавит вида А, Б, … , Ю, Я, а, б, …, ю, я. Также заполняется словарь frequencies, содержащий частоты появления русских букв в языке (данные взяты с https://ru.wikipedia.org/wiki/Частотность). Выводится алфавит с номерами всех букв в 1-индексации. Выводится мощность алфавита. Создаются словари, содержащие алфавиты только малых букв (lowered_alphabet_liter_number и lowered_alphabet_number_liter). Выводится предупреждение, что далее все буквы в считанном тексте переводятся в нижний регистр. Пользователь вводит путь к файлу формата .txt, программа считывает её в строку path и передаёт

функции reading_text – аналогу reading, но для считывания обычного текста, а не бинарных данных. Полученный текст переводится в нижний регистр (метод .lower()). Пользователю предлагается вывести считанный текст. Далее, выводится таблица частотностей букв русского алфавита. Приложение спрашивает пользователя: «Содержит ли ваш текст большое количество необщеупотребительной лексики?». В случае положительного ответа выводится предупреждение о том, что анализ может ошибиться с большой вероятностью. Иначе программа заверяет пользователя, что алгоритм, скорее всего, вернёт верный ответ. Далее, применяется алгоритм частотного анализа. После получения ответа пользователю предлагается вывести расшифрованный текст в консоль. Приложение спрашивает, что делать с полученным текстом: 0 — ничего, 1 — перезаписать в переданный изначально файл, 2 — записать в какой-то другой файл (формата .txt) вместо имеющегося там содержимого. Далее, вызывается функция writing_text (аналог writing для работы с содержимым .txt-файлов). Выводится мини-отчёт — количество символов в обработанном тексте (включая не входящие в алфавит — такие символы не обрабатываются и не изменяются), предполагаемый ключ (optimal_key – значение, которое алгоритм посчитал наиболее подходящим для расшифровки) и среднее отклонение от ожидаемого частотного распределения.

После выполнения основной части программа предлагает пользователю продолжить работу. В случае отрицательного ответа программа завершается.

Алгоритмическая сторона вопроса.

1.1. Шифр Цезаря. К считанной байтовой последовательности применяется сдвиг shift_pattern, т. е. если байт имеет значение value, то он заменяется на (value + shift_pattern) % n, где n = 256.

При дешифровке с известным ключом shift_pattern применяется, фактически, алгоритм шифрования, но уже с ключом (-shift_pattern), т. е. value заменяется на (value – shift_pattern) % n, где n = 256.

*1.2. Шифр Вижинера***.** Фактически, усовершенствованный шифр Цезаря. Сначала размеры «байтового» текста и ключевого слова выравниваются (меньшее увеличивается за счёт прибавления слева копии самого себя (возможно, неполной копии, если она - последняя прибавляемая) до размеров большего). Затем для каждого j из отрезка [1; max(размер байтового текста, размер ключевого слова)] к j-му элементу байтового текста применяется сдвиг shift_j, где shift_j равен порядковому номеру в алфавите той буквы, которая стоит на j-ой позиции ключевого слова key (т. е., фактически, к j-му элементу байтового слова применяется шифрование Цезаря с ключом shift_j). Пользователю возвращается байтовое слово, состоящее из первых first_text_len байтов слова, полученного в результате выполнения алгоритма (во время работы алгоритма слово могло быть увеличено, но к изначальному тексту имеют отношение лишь первые first_test_len букв), где first_text_len – длина байтового текста, изначально считанного из файла.

Дешифрование эквивалентно шифрованию по ключу, «обратному» данному — т. е. если j-ая буква ключевого слова key имеет номер t в алфавите, то она заменяется на (255 — t)-ую букву алфавита.

*1.3. Шифр Вернама***.** Фактически — усовершенствованный шифр Вижинера, только теперь ключевое слово не вводится пользователем, а генерируется из случайных (псевдослучайных) букв алфавита, причём сразу размера, равного размеру текста. Далее применяется шифрование Вижинера.

Дешифрование эквивалентно дешифрованию шифра Вижинера.

*2. Взлом шифра Цезаря***.** В программе из алгоритмов взлома представлен один — взлом шифра Цезаря методом частотного анализа. В программе это единственный алгоритм, применяющийся только к строкам (эксперименты показали, что применение данного метода к дешифрованию байтовых текстов бесполезно).

До выполнения основного цикла составляется таблица частотности букв (в программе выбран русский алфавит) frequencies вида (буква; частотность

буквы в русском языке). Далее, считывается текст, и все его буквы переводятся в нижний регистр для значительного упрощения подсчёта частотности букв в введённом тексте.

Затем запускается цикл по k от 0 до 255, и для каждого k составляется таблица частотности букв в введённом слове (в программе это словарь new_frequencies вида (буква a; n_a / n, где n – длина текста без учёта символов, не входящих в алфавит (они не изменяются), n_a – количество вхождений буквы a в текст) при сдвиге слова на k. Для каждой такой таблицы подсчитывается сумма побуквенных отклонений new_frequencies от frequencies – суммарное отклонение и среднее буквенное отклонение sum_delta . С помощью такого цикла находится optimal_k – сдвиг, при котором sum_delta минимально, т. е. , скорее всего, при применении алгоритма шифра Цезаря к тексту с ключом optimal_k, будет получено зашифрованное сообщение. Алгоритм возвращает optimal_k и предполагаемое расшифрованное сообщение.

При этом, программа предупреждает пользователя, что если текст содержит малое количество алфавитных символов или большое количество необщеупотребительной (например, профессиональной морской) лексики, то велика вероятность ошибки (например, в профессиональной морской лексике намного чаще, чем в общеупотребительном языке, встречается буква «ф» в силу большого числа заимствований из иностранных языков).

Пояснение к отдельным переменным и функциям. Большая часть переменных и функций, использованных в программе, названы так, чтобы смысл их действий, аргументов, возвращаемых значений был интуитивно понятен читающему. Кроме того, смысл части переменных и функций объяснён в предыдущих разделах. Однако, следует дать некоторые комментарии.

1) Переменные с названиями users_answer – переменные-флаги, считывающие ответы пользователя на некоторые вопросы программы.

2) Ряд функций имеют функции-дублёры, отличающиеся суффиксом _text в названии. Функции с таким суффиксом имеют тот же смысл, что и

«бессуфиксальные», но применяются только в алгоритме частотного анализа для обработки строк.

3) функция caesar_cipher_without_talks — аналог функции caesar_cipher, предназначенный для работы внутри алгоритма частотного анализа, без непосредственного взаимодействия с пользователем.

4) ряд функций и переменных имеют в названиях слова «text», «word», но не относятся к условию, выполняемому при mode == 3 и к функциям, содержащим суффикс _text, т. е. не имеют отношения к алгоритму частотного анализа. Такие переменные и методы нужны для алгоритмов шифрования и дешифрования файлов, и, как правило, содержат типы bytes и bytearray.

5) основные функции имеют вид <фамилия учёного>_chipher(). Они являются основными потому, что в них и в вызываемых ими функциях происходит шифрование и вывод результатов. Среди их аргументов встречается bool-переменная flag. Она нужна для отделения шифрования от дешифрования — flag == True, если функция вызвана для шифрования, иначе функция вызвана для дешифрования.