**ОТЧЁТ**

о выполнении практической работы по теме:

«Анализ возможностей по обращению мутации слов»

Выполнил:

Лепёшкин К.С.\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Москва

2020 г.

Оглавление

[Введение 3](#_Toc40746909)

[1 Задачи проекта 4](#_Toc40746910)

[2 Используемые инструменты 5](#_Toc40746911)

[2.1 Выбор языка программирования 5](#_Toc40746912)

[2.2 Выбор алгоритма для выделения лексем из текста без пробелов 5](#_Toc40746913)

[2.3 Выбор алгоритмов для исправления некорректных слов 5](#_Toc40746914)

[2.4 Выбор алгоритма для выделения базовых частей слова 7](#_Toc40746915)

[2.5 Выбор библиотеки для тестирования 7](#_Toc40746916)

[3 Разработка программного средства 8](#_Toc40746917)

[3.1 Структура проекта 8](#_Toc40746918)

[3.2 Основной модуль 9](#_Toc40746919)

[3.3 Модуль конфигуратора 10](#_Toc40746920)

[3.4 Модуль загрузки списка слов 12](#_Toc40746921)

# Введение

Основная цель данной работы заключается в определении возможностей по обращению мутаций слов с последующим написанием программного средства, способного обращать данные мутации.

В чём же состоит проблема и насколько она актуальна? Почти каждый продвинутый пользователь осведомлен о существовании программного обеспечения, способного мутировать слова и текст для определенных целей. Одной из основных целей является вполне обычное желание каждого пользователя защитить собственную информацию. Как раз для таких целей создано множество утилит, как например Hashcat, способный мутировать слова по любому шаблону, а также открытое программное обеспечение, созданное пользователями Сети и располагающееся в открытом доступе на многих git-ресурсах, к примеру, Github.

Но если что-то можно зашифровать, то вполне вероятно, это можно и расшифровать, то есть получить определенный набор слов, к которому мы можем применить те же самые методы мутации и получить наше зашифрованное слово. Основная проблема в том, что проверить данное утверждение в нашем случае затруднительно, так как доступного рядовому пользователю программного обеспечения найти не представляется возможным. Что это значит? Единственная технология, которая удовлетворила бы наши задачи, имеется у коммерческих организаций, как например, ElcomSoft, создающие приложения, способные восстанавливать пароли, просматривать скрытую информацию операционной системы и выполнять множество других задач, связанные с обеспечением безопасности данных. Но, как можно понять, данное программное обеспечение является коммерческим.

Имеется ли возможность найти открытую технологию, которая удовлетворила бы нашим требованиям? С большой вероятностью, ответ отрицательный.

Таким образом, цель данной работы заключается в написании программного обеспечения, способного получить определенный набор слов, которые являются базовыми словами, то есть теми, которые были мутированы. Или же доказать, что получить такой набор слов не представляется возможным.

# Задачи проекта

Мы определили главную проблему в том, что доступного программного обеспечения, удовлетворяющее нашим требованиям, для выполнения цели не существует.

Таким образом, чтобы реализовать данное приложение, способное осуществить поставленную цель, следует рассмотреть ряд задач, которые будут выполнены в ходе данной практической работы.

Всего мы можем выделить 6 основных задач:

1. проанализировать существующие методы выделения лексем из текста без пробелов, чтобы иметь возможность корректно выделять базовые части слова и исправлять ошибки по мере необходимости;
2. реализовать и протестировать методы выделения лексем с удалением некорректных символов, чтобы иметь возможность очищать слова от некорректных символов;
3. проанализировать существующие методы выделения базовых частей из слова, полученного путем объединения несвязных слов, чтобы иметь возможность составить минимальный набор слов, удовлетворяющий нашей цели;
4. реализовать и протестировать метод выделения базовых частей из слова, чтобы составить набор базовых слов;
5. измерить время работы реализованных методов на большой выборке данных, чтобы получить представление о производительности программы и иметь возможность увеличения данной производительности;
6. реализация и тестирование программного средства сравнения совпадений в заданных файлах, чтобы иметь возможность сопоставить полученный набор слов с исходными задуманными словами.

В ходе выполнения поставленных задач мы сможем понять насколько поставленные цели осуществимы.

# Используемые инструменты

## Выбор языка программирования

Для реализации задач, поставленных в предыдущем пункте, мной были проанализированы существующие программные средства, являющиеся наиболее целесообразными для конкретной задачи. Целесообразность программного средства заключается в простоте его использования, чётком выполнении задачи, которую мы ставим перед этой программой, а также в актуальности на данный момент времени.

Все выбранные инструменты будут использоваться для языка программирования Python 3.7, поскольку данный язык обеспечивает высокую скорость разработки, совместимость со многими высокоуровневыми языками, что позволяет реализовывать программы на более производительном языке и внедрять их в свои модули, а также множество встроенных инструментов для огромного спектра задач.

Выбранные программные средства можно разделить на 4 группы в соответствие с выполнением определённой задачи.

## Выбор алгоритма для выделения лексем из текста без пробелов

Данный алгоритм был найден на сервисе GitHub и является открытым программным средством. Поскольку он не удовлетворял задаче выделения лексем с достаточным быстродействием, он был изменён и приведен к окончательному виду.

Данный алгоритм реализован с применением метода динамического программирования и использует так называемую «стоимость» слова, то есть математически вычисляемую величину, прямо пропорциональную рангу слова в списке слов, упорядоченных по частоте употребления.

## Выбор алгоритмов для исправления некорректных слов

Для начала уточним, что подразумевается под некорректным словом. Слово является некорректным, если при написании этого слова используются цифры, служебные символы, пробельные символы и тому подобное (например, 123password), если в написанном слове содержится орфографическая ошибка (например, pasword, password) А также «Leet» символы, то есть символы, полученные заменой латинских букв на похожие цифры и символы (например, p4s5w0rd).

Поскольку, невозможно найти готовый алгоритм, который способен был бы решить решения проблемы с «Leet» символами, а также другими некорректными символами, мной был написан собственный алгоритм, реализованный на основе подсчёта «стоимости» слова.

Для решения задачи по исправлению орфографических ошибок я воспользовался известным алгоритмом БК-дерева. Данный алгоритм выстраивает дерево, основываясь на определенном расстоянии между узлами. Узлами данного дерева выступают слова из частотного списка. А в качестве расстояния между узлами мной было выбрано расстояние Дамерау-Левенштейна, поскольку оно идеально подходит для задач нахождения близких слов (то есть слов, которые минимально отличаются друг от друг посимвольно). Расстояние Дамерау-Левенштейна подсчитывает расстояние, учитывая не только взаимное расположение букв в словах, но и их перестановки (это улучшенное расстояние Левенштейна).

Для реализации БК-дерева мной был создан собственный класс, являющийся обёрткой для уже реализованного алгоритма дерева, находящегося в открытом доступе на ресурсе GitHub. Пакет, который я выбрал называется «pybktree». Поскольку этот пакет является актуальным и поддерживается разработчиком на данный момент, а также предоставляет понятный интерфейс для использования, было принято решение использовать данный вариант. Сам алгоритм мной изменён не был.

Расстояние Дамерау-Левенштейна реализовано также сторонним разработчиком и является открытым программным обеспечением. Пакет, который мной будет использоваться, называется «pyxdameraulevenshtein». Мой выбор пал на данный пакет потому, что он является лучшей реализацией данного алгоритма на данный момент. Алгоритм реализован с применением языка программирования Cython, упрощающий написание модулей С/С++ кода для Python. Код Cython преобразуется в С/С++ код для последующей компиляции и впоследствии может использоваться как расширение стандартного Python, что делает данный модуль производительнее любых других модулей, реализующих данный алгоритм, написанный на чистом Python.

Практический опыт использования пакета «pyxdameraulevenshtein», доказывает, что производительность увеличивается в более, чем 10 раз в сравнении с любым другим аналогом, написанным на Python.

## Выбор алгоритма для выделения базовых частей слова

Для выделения базовых частей слова мной будет применяться собственный алгоритм, реализованный на основе подсчёта «стоимости» слова.

После выделения базовых частей, я буду искать корень каждой части, поскольку нам необходимо получить минимальный словарь слов. Для того, чтобы получить корень слова, я буду применять технологию стемминга, реализованную в библиотеке «Natural Language Toolkit» (NLTK). Данная библиотека содержит пакет библиотек и программ для символьной и статистической обработки естественного языка, написанных на языке программирования Python. Поскольку она является быстрой для выполнения задачи стемминга, а также имеет простой в использовании интерфейс, мной была выбрана данная библиотека.

## Выбор библиотеки для тестирования

Для тестирования всех реализованных методов мной будет использоваться встроенный в язык Python модуль «unittest». Данный модуль предоставляет абсолютно весь необходимый для тестирования функционал, что позволяет тестировать модули быстро и эффективно.

Для реализации так называемых mock-текстов, мной был использован модуль «mock», входящий в состав «unittest».

# Разработка программного средства

Описание разработки данного программного средства будет включать в себя последовательное описание всех модулей и компонентов программы. В процессе описания работы модулей, будут соответственно выполняться задачи, описанные раннее, а также рассмотрены реализации всех алгоритмов, необходимые для достижения поставленной цели.

Основной идеей построения всего программного средство было в том, чтобы предоставить пользователю максимально возможный функционал для получения удовлетворяющего его результата. Поэтому, реализована идея выбора 4 режимов работы, способных работать одновременно: посчитать «стоимость» слова, очистить слово от некорректных символов, исправить слово и выделить базовые части слова.

Для предоставления более гибкого функционала, реализована возможность изменения 4 дополнительных параметров, которые напрямую влияют на работу методов обработки и исправления слов, что позволяет получить оптимальный результат.

Пользователю предоставляется возможность вывода результата в файл для каждого режима работы, а также в стандартный поток вывода. Пользователь может запросить более понятный для человека вывод с помощью определенной опции (опция –v). Однако, если пользователю необходимо получить только список слов, и ничего кроме, ему достаточно не указывать дополнительной опции.

## Структура проекта

Структура всего проекта состоит из пакетов «analyzer», в котором содержится все основные модули, «tests», в котором содержатся тесты для модулей, а также директории «data», содержащая списки слов, и сторонние конфигурационные файлы.

Структура проекта приведена рисунке 3.1.1.

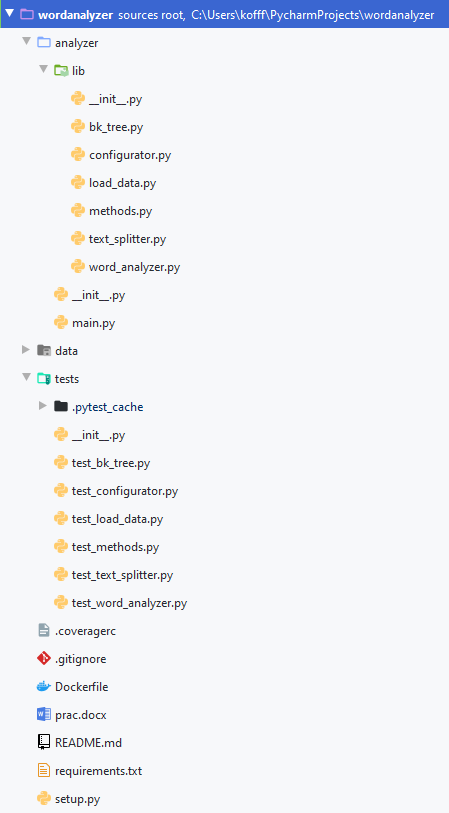


Рисунок 3.1.1 – Демонстрация структуры проекта

## Основной модуль

Основным модулем программы, с которого начинается старт является модуль main.py (См. Листинг 3.2.1).

Основная идея данного модуля состоит в том, чтобы, получив аргументы от пользователя из командной строки, передать их в функцию, которая обработает аргументы и выдаст определенный результат.

Для достижения данной задачи было принято решение создать класс Configurator, который обработает все аргументы командной строки и предоставит интерфейс для получения данных.

Классом, который будет выполнять все вычисления, является класс WordAnalyzer, принимающий на вход статической функции build объект класса Configurator. Благодаря внутреннему интерфейсу, класс WordAnalyzer получит необходимые данные и запустит функцию analyze(), выполняющая вычисления.

Листинг 3.2.1

import sys

from analyzer.lib.configurator import Configurator

from analyzer.lib.word\_analyzer import WordAnalyzer

def main():

configurator = Configurator(sys.argv[1:])

analyzer = WordAnalyzer.build(configurator)

analyzer.analyze()

if \_\_name\_\_ == '\_\_main\_\_':

main()

## Модуль конфигуратора

Данный модуль реализует интерфейс между получением аргументов из командной строки от пользователя и передачей этих данных другому объекту.

Чтобы правильно получить и обработать аргументы командной строки используется модуль Argparse, позволяющий задать аргументы, которые необходимо получить на вход программе.

Основная идея состоит в следующем: конструктор класса Configurator получает на вход аргументы командной строки. Данные аргументы, а также параметры описания, названия и другие параметры программы передаются в статический метод создания объекта класса ArgumentParser (класс из модуля Argparse). Метод возвращает созданный объект, с помощью которого в последствие будет осуществляться обработка аргументов (См. Рисунок 3.3.1).

Объект класса ArgumentParser обрабатывается методом \_get\_parameters (См. Рисунок 3.3.2), возвращающий объект, содержащий все переданные пользователем параметры. Важно отметить, что в данном методе производится проверка на наличие обязательных аргументов: исходного файла со списком слов, которые необходимо обработать или список слов, переданный напрямую; имя файла, содержащий частотный словарь; хотя бы один из режимов работы программы.

Используя полученный из метода \_get\_parameters объект, класс предоставляет свой API для предоставления данных. Каждый метод обрабатывает данные необходимым образом и предоставляет вывод.

Для загрузки списка слов в классе присутствует метод \_load\_words (См. Рисунок 3.3.2), являющимся обёрткой над внешней функцией load\_words из метода load\_data. Данный классовый метод обрабатывает возможные ошибки и возвращает загруженный список слов. Важно отметить, что если пользователь передает список слов напрямую из командной строки (опция –w), то даже если опция получения списка слов из файла (опция –s) присутствует, она будет проигнорирована, так как первая опция является приоритетной.

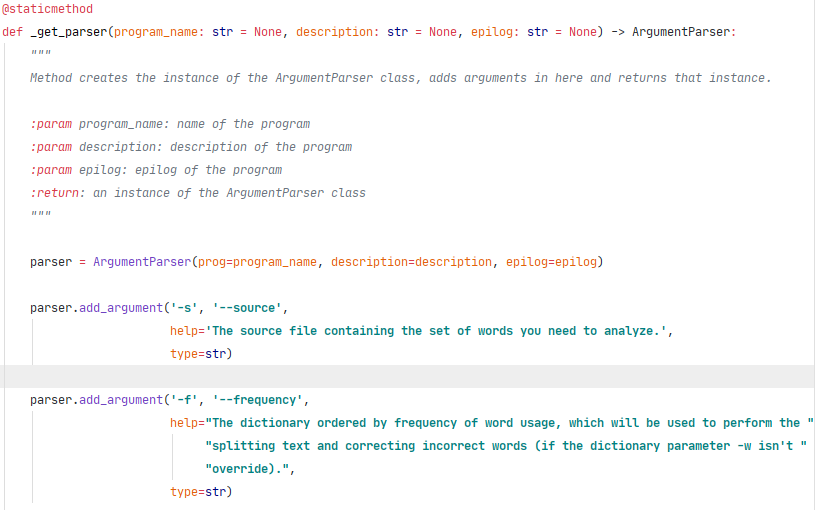


Рисунок 3.3.1 – Демонстрация метода создания объекта класса ArgumentParser



Рисунок 3.3.2 – Демонстрация методов получения параметров и загрузки слов

## Модуль загрузки списка слов

Предположим, что нам необходимо получить список слов из файла. Но что, если файл чрезмерно большой, а нам необходимо получить лишь случайный срез? А если кодировка файла нестандартная и файл не может быть просто обработан? Для этих ситуаций создан модуль load\_data, содержащий функцию load\_words.

Пользователь может передать функции определенное количество слов, которое ему необходимо обработать (опция –c). А если пользователю необходимо указать кодировку файла, он может передать её с помощью опции –e.

В данной функции обрабатываются стандартные ошибки пользователя, связанные с неверным вводом имя файла или случая, когда указан пустой файл.

Листинг данного модуля приведен на рисунке 3.4.1.

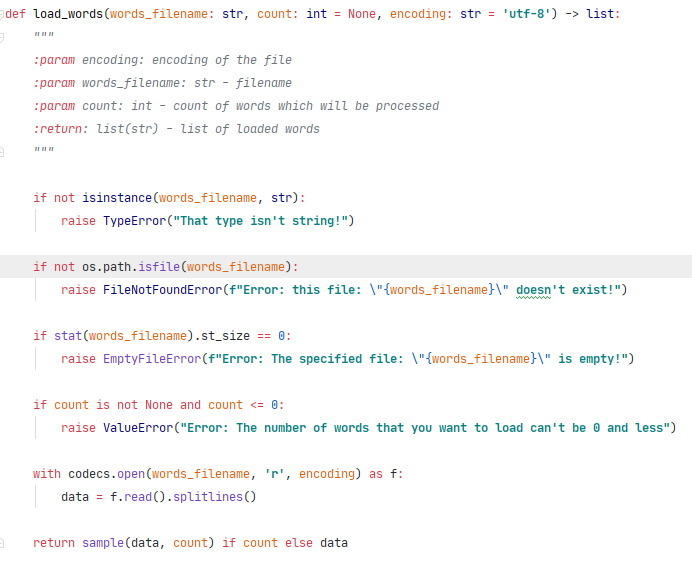


Рисунок 3.4.1 – Демонстрация основной функции модуля load\_data