Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский

политехнический университет»

(ПНИПУ)

Кафедра вычислительной математики и механики

**Лабораторная работа № 6**

по дисциплине:

«**Информационная безопасность и защита информации**»

по теме:

«Частотный криптоанализ»

Выполнил:

Студент группы ИСТ-19-1бзу

Соболь Е. В.

Проверил:

доцент кафедры ВММБ

Труфанов А. Н.

Пермь

2022 г.

**Цель работы:**

Ознакомиться с методами частотного криптоанализа, получить практические навыки создания ПО для криптоанализа зашифрованного текста.

# Описание алгоритма криптоанализа:

**Метод Цезаря**

Частотный анализ шифра цезаря - один из методов, применяемых при дешифровке. Он заключается в том, что для каждой буквы алфавита есть усредненная частота появления в тексте. Если взять том "Война и Мир", а затем посчитать количество всех букв "ж" в книге, потом отнормировать значение на 100 относительно числа всех букв книги, то получится число, приблизительно равное ~ 0,94. Подобные значения рассчитаны для каждой буквы многих алфавитов. В русском языке наиболее часто встречающейся буквой, с точки зрения статистики, является буква "о", с частотой 10,983. Чтобы понять, в чём заключается роль криптоанализа, опишем работу алгоритма взлома шифра цезаря:

1) Запросить исходный текст.

2) Найти средние частоты появления каждой буквы.

3) Найти смещения каждой буквы относительно эталонных.

4) Посчитать количество смещений.

5) Смещения с наибольшим количеством считать ключами.

6) Расшифровывать текст, используя полученный (ые) ключ(и).

Также задачу взлома шифра цезаря можно решить методом грубой силы, иногда, так задача решается намного быстрее и эффективнее.

# Метод Виженера

Метод Касиски предназначен для определения длины ключа шифра Виженера. Это дает возможность применить частотный анализ для независимого взлома каждого из подключей. Рассмотрим этапы алгоритма, которые будут реализованы данной программой взлома шифра Виженера:

* Нахождение повторяющихся сегментов.
* Определение множителей на интервалах повторения.
* Получение каждой n-й буквы строки.
* Применение частотного анализа для взлома подключей.
* Перебор возможных подключей методом грубой силы.

# Нахождение повторяющихся сегментов

В соответствии с методом Касиски первое, что необходимо сделать, - это найти в шифре повторяющиеся сегменты, содержащие, по крайней мере, три буквы. Такие последовательности могут быть повторениями одних и тех же наборов символов в исходном тексте, зашифрованных с помощью одних и тех же подключей. Например, если зашифровать открытый текст "ТНЕ САТ IS OUT OF ТНЕ BAG" с помощью ключа "SPILLTHEBEANS", то получим следующее:

****

Рисунок 1 – повторяющиеся сегменты

Сочетание "LWМ" повторяется дважды. Причина заключается в том, что группа букв "LWМ" в шифре - это слово "ТНЕ", зашифрованное с помощью одних и тех же букв ключа, "SPI", поскольку так получилось, что ключ повторяется на втором слове "ТНЕ". Количество букв от начала первого буквосочетания "LWM" до начала второго такого же буквосочетания, которое мы будем называть интервалом повторения, равно 13. Это дает основания полагать, что длина ключа, применяемого в данном шифре, составляет 13 букв.

Таким образом, мы выявили длину ключа путем простого анализа повторяющихся последовательностей.

Однако в большинстве шифров ключ не будет удобным образом выравниваться по повторяющимся последовательностям букв, или же ключ может встречаться между повторяющимися последовательностями более одного раза, т.е. интервал повторения будет кратным длине ключа, а не равным ей. Чтобы попытаться найти пути решения этой проблемы, рассмотрим более длинный пример, в котором нам неизвестно, что собой представляет ключ.

Если удалить из шифра "PPQCA XQVEKGYBNKMAZU YВNGBAL JON I TSZM JYIM. VRAG VOHT VRAU С TКSG.DDWUO XITLAZU VAVV RAZ С VКВ QP IWPOU" все небуквенные символы, то он примет вид строки, приведенной на рис. 2. На этом рисунке также выделены повторяющиеся последовательности - "VRA' , "AZU" и "YBN" - и указано количество букв между каждой парой последовательностей.

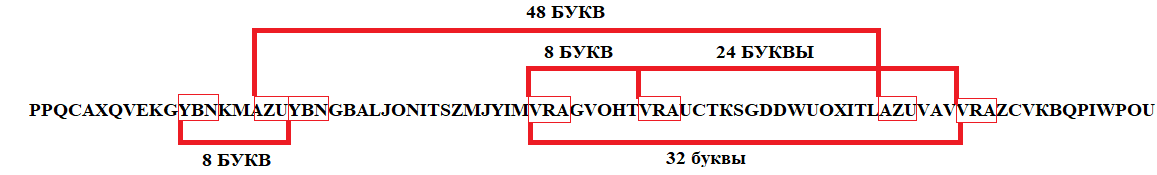


Рисунок 2 - Повторяющиеся последовательности в шифре

В этом примере существует несколько возможных вариантов длины ключа. Следующий шаг метода Касиски заключается в вычислении множителей всех интервалов повторения для сужения круга возможных вариантов.

# Определение множителей на интервалах повторения

В нашем случае промежутки между последовательностями, т.е. интервалы повторения, равны 8, 8, 24, 32 и 48. Но для нас важны не они, а их множители. Чтобы понять, почему это так, рассмотрим сообщение "THEDOGANDTHECAT" (рис. 3) и попытаемся зашифровать его с помощью девятибуквенного ключа ABCDEFGHI, а также трехбуквенного ключа XYZ. Каждый из этих ключей повторяется на протяжении всей длины сообщения.

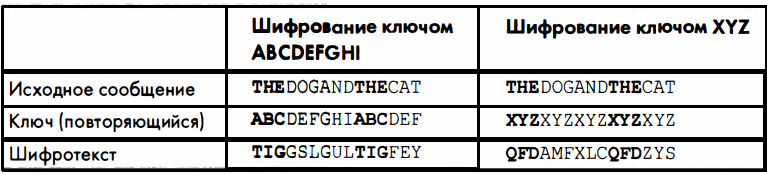


Рисунок 3 - Шифрование сообщения двумя различными ключами

Два ключа приводят к двум разным шифрам. Ни исходное сообщение, ни ключ не будут известны взломщику шифра, но он заметит, что в шифре "TIGGSLGULTIGFEY" последовательность "TIG" встречается при значениях индекса 0 и 9. Поскольку 9 – 0 = 9, интервал повторения для данной последовательности равен 9, а это может свидетельствовать о том, что истинный ключ является девятибуквенным, и в рассматриваемом случае он именно такой. Однако и в шифре "QFDAМFXLCQFDZYS" имеется своя повторяющаяся последовательность, "QFD", которая встречается при значениях индекса 0 и 9. Интервал повторения здесь тоже равен 9, и это, на первый взгляд, указывает на то, что ключ, использованный для получения данного шифра, насчитывает девять букв. Однако мы знаем, что в действительности ключ содержит всего три буквы: "XYZ". Повторяющиеся последовательности встречаются тогда, когда одни и те же буквы в исходном сообщении ("ТНЕ" в нашем примере) шифруются одними и теми же буквами ключа (в данном примере это "АВС" и "XYZ"), что происходит в том случае, если аналогичные группы в сообщении и ключе выравниваются и шифруются в одну и ту же последовательность. Подобное выравнивание может встречаться в позициях, кратных длине реального ключа (таких, как 3, 6, 9, 12 и т.д.), и именно поэтому трехбуквенный ключ может порождать последовательности с интервалом повторения, равным 9. Таким образом, возможная длина ключа может быть обусловлена не только интервалом повторения, но и его множителями. Множителями числа 9 являются числа 9, 3 и 1. Поэтому, если найдена последовательности с интервалом повторения 9, то необходимо рассмотреть два возможных варианта длины ключа: 9 и 3. Ключ длиной 1 можно игнорировать, поскольку шифр Виженера с ключом такой длины сводится к шифру Цезаря.

Второй этап метода Касиски включает нахождение множителей каждого из интервалов повторения (рис. 3, 4)

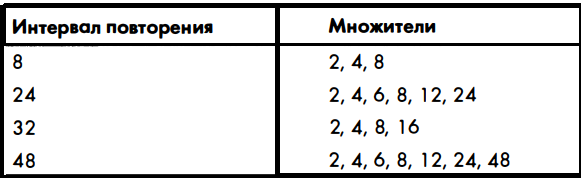


Рисунок 4 - Множители выявленных интервалов повторения

Числа 8, 8, 24, 32 и 48 в совокупности имеют следующие множители: 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 8, 8, 8, 8, 12, 12, 1 6, 24, 24 и 48. С наибольшей вероятностью ключами окажутся чаще всего встречающиеся множители. Поскольку в данном примере чаще остальных встречаются множители 2, 4 и 8, то именно они являются наиболее вероятными вариантами длины ключа Виженера.

# Получение каждой n-й буквы строки

Когда известны возможные значения длины ключа, можно воспользоваться этой информацией для дешифрования сообщения по одному подключу за раз. Предположим, что длина ключа в нашем примере равна 4. Если нам не удастся взломать шифр, мы попытаемся использовать ключи длиной 2 или 8. Поскольку ключ применяется циклично, использование ключа длиной 4 означает, что каждая четвертая буква открытого текста шифруется первым подключом, каждая четвертая буква исходного текста, начиная со второй, шифруется вторым подключом и т.д. Сформируем строки из букв шифра, шифруемых одним и тем же подключом. Сначала мы определим, какой будет каждая четвертая буква строки, если начинать с разных букв, а затем объединим все такие буквы в одну строку. В рассматриваемых примерах каждая четвертая буква выделяется полужирным шрифтом. Идентифицируем каждую четвертую букву, начиная с первой:



Рисунок 5 – Чередование с начала

Далее находим каждую четвертую букву, начиная со второй:



Рисунок 6 – чередование со сдвигом

Затем делаем то же самое, начиная с третьей и четвертой буквы, пока не исчерпаем всю проверяемую длину ключа. На рис. 7 приведены объединенные строки, составленные из букв, выделенных полужирным шрифтом на каждой итерации.

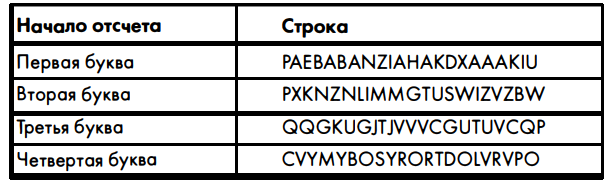


Рисунок 7 - Строки, составленные из каждой четвертой буквы

# Применение частотного анализа для взлома подключей

Если была угадана длина ключа, то каждая из четырех строк, созданных в предыдущем разделе, должна быть зашифрована с помощью одного ключа. Это означает, что если строка, зашифрованная корректным ключом, подвергается частотному анализу, то дешифрованные буквы, вероятнее всего, будут иметь высокую оценку частотного соответствия.

Рассмотрим, используя в качестве примера первую строку: PAEBABANZIAHAKDXAAAKIU. Дешифруем строку 26 раз, по одному разу для каждого из 26 возможных подключей, используя функцию vigene re\_сipher.decryptMessage() из лабораторной работы № 4. Затем протестируем каждую дешифрованную строку, используя функцию частотного анализа для английского языка freqAnalysis.englishFreqMatchScore() (см. листинг программы).

Результаты представлены на рис. 8, 9

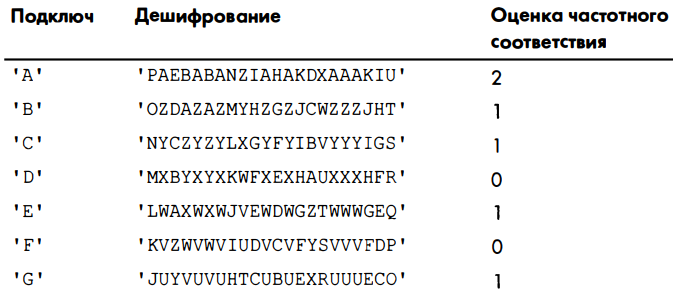


Рисунок 8 - Оценки частотного соответствия для каждого варианта дешифрования

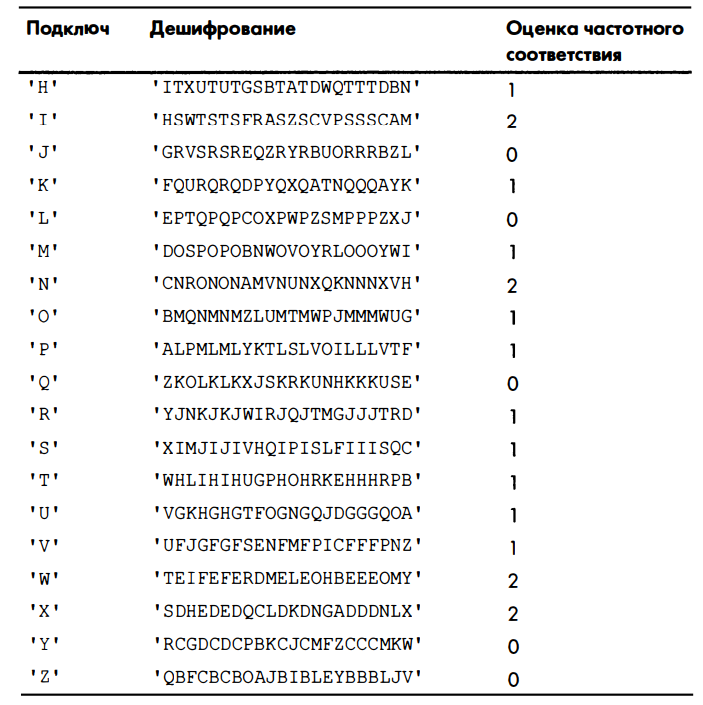


Рисунок 9 - Оценки частотного соответствия для каждого варианта дешифрования (продолжение)

Подключи, которые приводят к получению вариантов дешифрования с наибольшей оценкой, являются самыми вероятными кандидатами на роль реального ключа. На рис. 8,9 подключи “А”, “I”, “N” , “W” и “Х” дают наивысшую оценку частотного соответствия для первой строки. Оценки в целом имеют низкие значения, поскольку объем шифра не позволяет получить достаточно длинную тестовую строку, но для данного примера этого вполне достаточно.

Следующий шаг заключается в повторении этого процесса для остальных трех строк с целью выявления наиболее вероятных вариантов подключей. Окончательные результаты приведены на рис. 10.

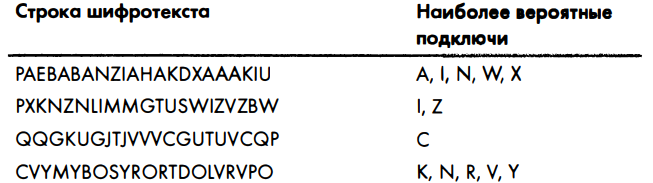


Рисунок 10 - Наиболее вероятные подключи для тестируемых строк

Поскольку для первой строки существуют пять возможных подключей, для второй - два, для третьей - один и для четвертой - пять, общее количество возможных комбинаций составляет 50 = 5 \* 2 \* 1 \* 5. Получается, нам нужно перебрать 50 возможных ключей. Это намного лучше, чем полный перебор 26 \* 26 \* 26 \* 26 = 456 976 возможных ключей. Разница становится еще более существенной в случае ключей Виженера большей длины.

# Перебор возможных подключей методом грубой силы

Метод грубой силы предполагает полный перебор всех возможных комбинаций подключей. Все 50 возможных комбинаций приведены на рис. 11.



Рисунок 11 - 50 возможных комбинаций

Завершающим этапом алгоритма реализуемого взлома шифра Виженера будет тестирование всех 50 возможных вариантов дешифрования на полном шифре, чтобы увидеть, какие из них приводят к получению осмысленного текста. Осмысленность проверяется с помощью модуля detectWords и словарей. В ходе этого процесса можно будет обнаружить, что ключом для дешифрования шифра "PPQCA XQVEKG…" является “WICК”.

**Гистограммы частоты встречаемости символов и наиболее встречаемых биграмм в тексте**

Для анализа было взято произведение Михаила Булгакова “Пропавший глаз”.

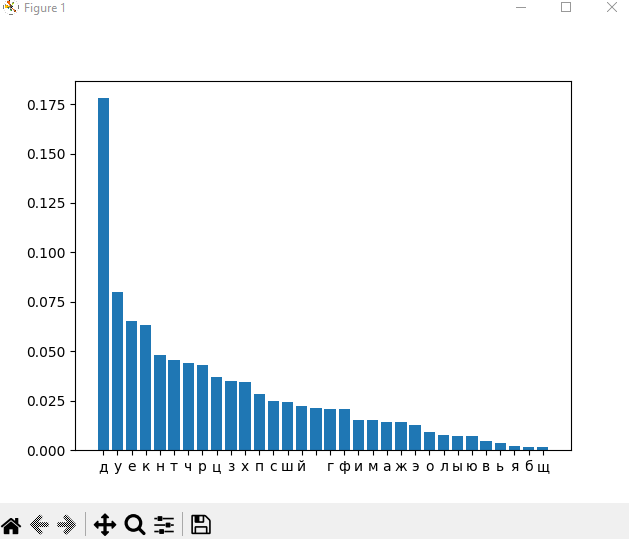


Рисунок 12 – Гистограмма встречаемости букв (зашифрованном)

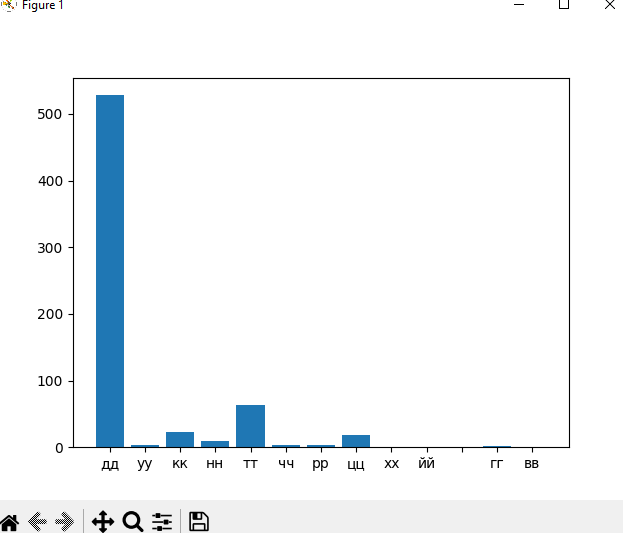


Рисунок 13 – Гистограмма встречаемых биграмм (зашифрованном)

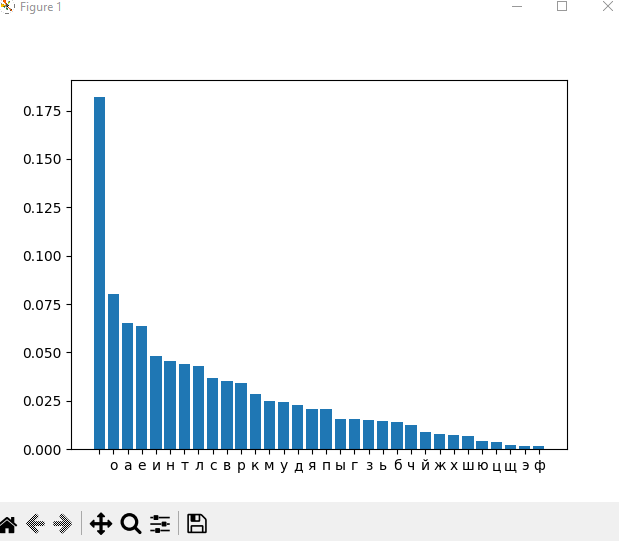


Рисунок 14 - Гистограмма встречаемости букв (расшифрованный)

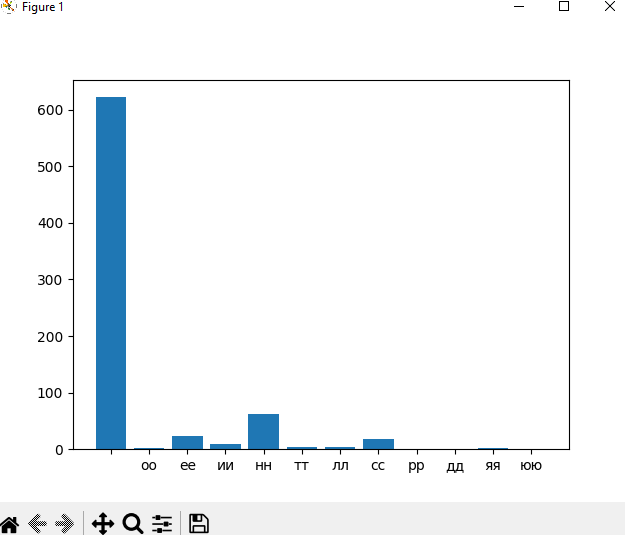


Рисунок 15 - Гистограмма встречаемых биграмм (расшифрованный)

**Фрагменты текста для метода Цезаря:**

Текст при шифровании приведён к одному регистру. Зашифрованный фрагмент:

сиыиущгфд йизиэыфгбйф,иуйуилилчщчяоиьщзшдзияцгуйфихйфдасбуй.

иииииучцоацч,иэицомчичуйрйфйыдишощофчхфоццйзицчмй,исилчьинлйиайыйихгиыиюофднбощчхилчрсфсыд,ицйуфйнглйзимсшычлэжишчлзруэицйихйфдасбуэ,иучьчщгтилгфишчнщзнинлйиайый.ишчьчхичконйьдицэпцчикгфч,ишчьчхифоцдикгфчикщсьдыз,иячьофчыдиаьч-цскэндишчасьйьд,ийиьйхишщсшчфрфсиыэхощус,ирйьзцэфчинйфс,исиз,иыучщкцчихчщвйыд,инчкщсфыз.ицчиьйуиуйуирэкайьгти«псффоь»ишщчфопйфишчрйкгьгхилихгфдцчтилчнои–ицйицохицйлоусичыьйфйыдищпйлоцдуйзишчфчычауй,иуйуишйхзьдичилоыоццсяищчнйяиэихчыьй.

ииииинй…икщсьдызиьщсищйрйилицонофжикгфчицсиуиаохэ.ишчщчжицйыирйцчысфчилчлыоиыцомчх,илгфйицоыэылоьцйзихоьофд,ихгишчинлйинцзиыснофсилихэщдолыучтикчфдцс о,ицоишчыгфйфсинйпоилилчрцоыоцыуирйинолзьдилощыьирйимйроьйхс,исинчфмсхсилоаощйхсизихощсфисихощсфиылчтиуйксцоьисипйнцчиячьофимйроь,иьйуипйнцч,иуйуилиноьыьлоипйпнйфиуэшощчлыучмчи«ыфончшгьй».ицчилыоипоийцмфстыусоирйхйбусицоишчьэяфсилчлыоицйихэщдолыучхицочксьйохчхичыьщчло,исилщохзичьилщохоцсизилгцсхйфисриаощцчмчиюэьфзщасуйикфоыьзвэжисмщэбуэисилзфчикщсфыз,илгячнсфимфйнустисиасыьгт,иуйуимчщнгтичыьщчлсьзцсц.ипйфдифсбд,иаьчицоучхэикгфчишчфжкчлйьдызицйихоцз.

ииииишчрлчфдьо…инй…илондикгфисиовоиыфэайт,иучмнй,ишчхцсьыз,илгцэфикщсьлэ,исиьчфдучиаьчийуысцдзишщсцоыфйилиуйксцоьилгвощкфоццэжиущэпуэиыиусшзьучх,иуйуилинлощдимщчрцчирйыьэайфсисилгрлйфсихоцз.исихгиыишофймоотислйцчлцчтиэояйфсилиыьщйбцэжинйфд,ирйуэьйццгоиликйщйцдсиьэфэшг,ишщчцоыфсыд,иуйуиаощцгтишщсрщйу,иычыьчзвстисриучцот,иуэаощйисицйы,иыулчрдилркоыслбстызикофгтичуойц.илджмйиылсыьофй,иуйуилондхй,илгфй,ишфолйфйыд,иячячьйфй,илыоиуиаощьэисыаорфч,исизисышгьглйфирцйучхчоишчячфчнйцсоимно-ьчиличкфйыьсиычфцоацчмчиышфоьоцсзишщсихгыфс,иаьчиычкдохызихгиыишэьсилиеьчтиыйьйцсцыучтилощьзвотызихмфоисишщчшйнохирйицчадилыо:исишофймозислйцчлцй,исиуэаощ,исифчбйнс,исиз.иово,ишчхцж,илчрцсуфйиэихоцзинэщй уйзихгыфдичиьчх,иаьч,иучмнйихгикэнохирйхощрйьдисилчьицйыицйшчфчлсцэирйцоыоьиыцомчх,изисийуэбощуо,исиыоко,исиуэаощэилшщгыцэихчщюст…ирйаох?..ийиьйу,иаьчкгицоихэасьдыз…и«рйхощрцобдиьг,ифоуйщд,исикорихчщюсзишщолчыячнцотбсхичкщйрчх, –ишчхцсьыз,ичьлоайфихцоиыэячтисирнчщчлгтимчфчы, –ицсбьчиьоко…»иэ-мэ-мэ!..ияй-ыыы!.. –иылсыьйфйилондхй,исицйыихчьйфч,ихчьйфчилиыйцзя…ицэ,ицйшоайьйжьиьйхилиыьчфсацчтимйроьоицйирйнцотиыьщйцс о,иаьчилчь,ихчф,иьйуисиьйу,ишчмскфсишщсисышчфцоцссиыфэпокцгяичкзрйццчыьотифоуйщдиьйучт-ьч,ийищйлцчишофймозислйцчлцйиыиуэаощчхисишйщчжиучцот.ихсщисяишщйяэилиыцопцчхихчщо.иьдюэ…иаьчилимчфчлэифороь,иучмнйиьокзиьйуицйрглйохгтинчфмиыфэпкгицоыоьисицоыоь…

Расшифрованный результат:

и с крыльца я услышал, как в ворохе тряпья хныкал мальчишка.ц конечно, у него оказалась переломленная нога, и вот два часа мы с фельдшером возились, накладывая гипсовую повязку на мальчишку, который выл подряд два часа. потом обедать нужно было, потом лень было бриться, хотелось что-нибудь почитать, а там приползли сумерки, затянуло дали, и я, скорбно морщась, добрился. но так как зубчатый «жиллет» пролежал позабытым в мыльной воде – на нем навеки осталась ржавенькая полосочка, как память о весенних родах у моста.ц да… бриться три раза в неделю было ни к чему. порою нас заносило вовсе снегом, выла несусветная метель, мы по два дня сидели в мурьевской больнице, не посылали даже в вознесенск за девять верст за газетами, и долгими вечерами я мерил и мерил свой кабинет и жадно хотел газет, так жадно, как в детстве жаждал куперовского «следопыта». но все же английские замашки не потухли вовсе на мурьевском необитаемом острове, и время от времени я вынимал из черного футлярчика блестящую игрушку и вяло брился, выходил гладкий и чистый, как гордый островитянин. жаль лишь, что некому было полюбоваться на меня.ц позвольте… да… ведь был и еще случай, когда, помнится, вынул бритву, и только что аксинья принесла в кабинет выщербленную кружку с кипятком, как в дверь грозно застучали и вызвали меня. и мы с пелагеей ивановной уехали в страшную даль, закутанные в бараньи тулупы, пронеслись, как черный призрак, состоящий из коней, кучера и нас, сквозь взбесившийся белый океан. вьюга свистела, как ведьма, выла, плевалась, хохотала, все к черту исчезло, и я испытывал знакомое похолодание где-то в области солнечного сплетения при мысли, что собьемся мы с пути в этой сатанинской вертящейся мгле и пропадем за ночь все: и пелагея ивановна, и кучер, и лошади, и я. еще, помню, возникла у меня дурацкая мысль о том, что, когда мы будем замерзать и вот нас наполовину занесет снегом, я и акушерке, и себе, и кучеру впрысну морфий… зачем?.. а так, чтобы не мучиться… «замерзнешь ты, лекарь, и без морфия превосходнейшим образом, – помнится, отвечал мне сухой и здоровый голос, – ништо тебе…» у-гу-гу!.. ха-ссс!.. – свистала ведьма, и нас мотало, мотало в санях… ну, напечатают там в столичной газете на задней странице, что вот, мол, так и так, погибли при исполнении служебных обязанностей лекарь такой-то, а равно пелагея ивановна с кучером и парою коней. мир их праху в снежном море. тьфу… что в голову лезет, когда тебя так называемый долг службы несет и несет…ц

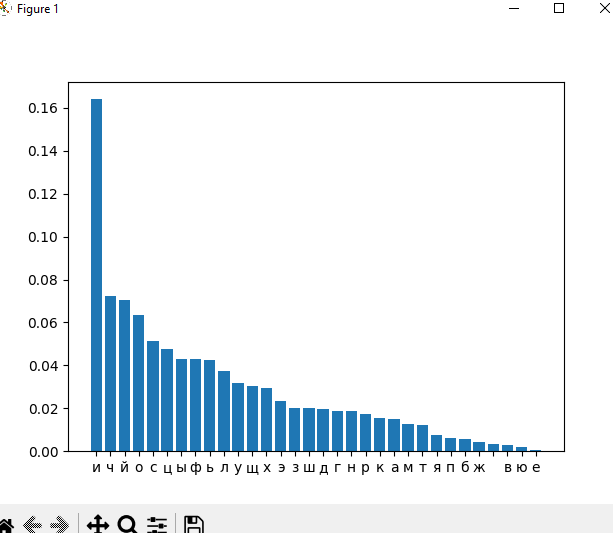


Рисунок 16 - Гистограмма встречаемых букв (зашифрованном фрагменте)

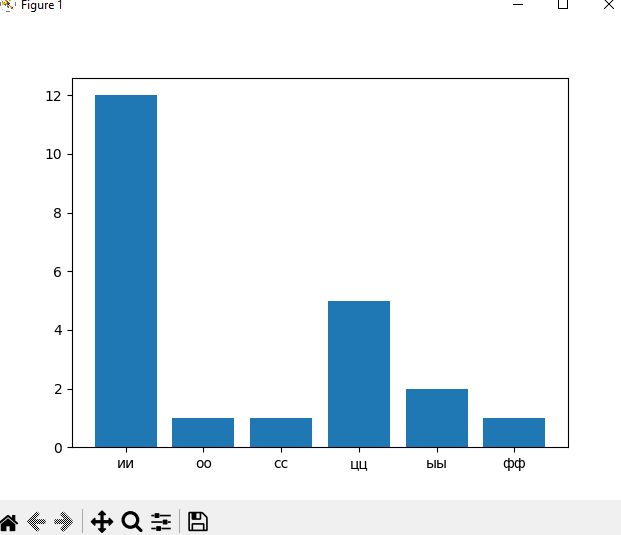


Рисунок 17 - Гистограмма встречаемых биграмм (зашифрованном фрагменте)

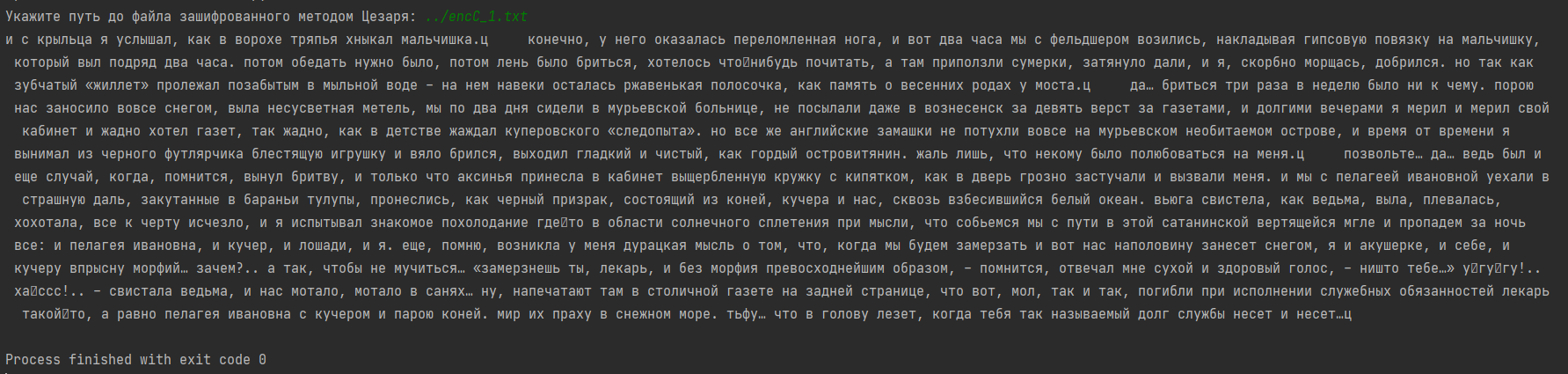


Рисунок 18 – скриншот работы программы caesarsHacker.py

**Фрагменты текста для метода Виженера:**

Текст при шифровании приведён к одному регистру. Зашифрованный фрагмент :

и т муымюща а хфльъгл, лвн в груоцж храсяя цпюкбн памюъищмг.

кппзчор, ц нёес олвкамвфь ржуемрплёпраа псгб, к еоу ёеа швфа нэ ф фёнядщжуон дсзйнлсэ, пгкмвжыгвв гйсфогхб ппдвзлх ра нвоьшкыкф, мстптюй гэо ппёуяе ёеа швфа. ррхон рдеевхь охйнп гюлп, сстпо оеою дымр дрйфяса, чстёнссэ щхо-окдуею тошкхаую, г тбо трйсслинл сфозрлк, каубрумр жамк, л я, тмсрвпс мптьатю, жовтллтб. ро увн кбм кувщгтьл «йимнзт» ртслёигл рркавэхын д пымюрок дсдё – пг нёо рагжни пухамвфь сигвёпякбб томрфошмг, кбм танбхь п дзсёприц тсдбч ц мпуха.

ев… дрйфяса фуи свка г пздёнб бьнс нй м ъенх. тосрб нбу каорфимр еогуз сожёон, дюлб пзсфуееупгя нжхемю, пы рр жвб ёря ткжемк е мфтяегунок гслэплцё, пз ппуюлбнл дбиз в гркнёузнтм ка ежеяую еесух зб егзёфгмй, к жомелмй дзчётгмй б песко и нжуим уеок мгбйпзт й игдор шоужо гбйзт, увн жбёро, лвн в ежхсудз жбижам мцпётсвтмсгп «уоеертыув». ро гуз жё вргмкмслкз збогшлк ре ррхуцнл впдфе ов пусюзвтмсм ожсбйфгенрп отфуогж, л всжпя пф ерёознй б еыокпам кк чётродр чуунвршкна внзсубьуя кёрфъну й двлп гуимув, вьчсдйн ёлбёник к ъитфюй, лвн гптжык рфтсреиубрио. иглэ нлшэ, щхо ожнонх дымр томадогвхьтб ра нжря.

ррквпнятё… ёг… вёёя бьн л еъж флфщгй, лрёдб, ссмокхса, дюнфн дрйфеу, й фслэмс чур гкткрьа суиожфлб д навкреу дющётдлёпруя муузмц с лктяумсм, лвн в едзрэ еуоипс збухушвои й дюзгвои нжря. й ою с ржоаджзй йдгнпдрок хзхбнл в тфуащпцю евоь, ивнууврньж е ббтгнэк хумхты, ртснёуоитю, нал щзроэм псккрбм, фотфсяъкм ии мснёл, нушжуа й пгс, тмеоию езвжфигълйтб демэм олжгн. гюбгб уеитфзлб, мгк гжжьнв, еымв, тлёдглбуя, хпчстбнг, втж н чётху йуъеинс, и а кфпьфювбн кнбмсмпж тоцрооевриё еже-ур е овнгсук фомпзчорёо тсоеужриа суи нэфлй, щхо трдьёофя нэ ф пффл в юфсй твхаокрслрм вётхяъжмса оёлё к трпсгдёо ка оръь гуз: и ржоаджв игврогпг, и лхъес, к оощвжи, й б. зщё, ссмоа, еоиплкмв ц мёпв дфтгцлвв мьуоь п фсм, шфс, кпежа нэ дуежп збозривхь й дст овф нбсслпдлнф йгнёузт тпзгпо, в и бмцшётне, й узбё, к нушжуу гсуытпц мптчик… йгчёо?.. г тбм, ътпгю нё оцчйфяса… «йгмёткнёъя ть, нзкбтя, и вжк мптчиа суегрфхпёрекълм пгуаирп, – ппориуув, оудзчбн пнё уцхпл л зеруогэм гпнсс, – окытп фзбё…» х-ёу-дх!.. шатуф!..– сгкфтбнг вёёямб, к рат остбнс, мпфглп д фаобш… нф, пгпёщгтбах тбо е суроишпсй двкеуж ра ивжнёл фтсвричж, ътп дст, нро, тбм л тбм, тодкдлй суи йутомпзнйк флфизбоэш овбкаопссужм лёмгрэ фгкпл-хо, б тгвор темвёеа кеаоренб у нушжуон к тасрб кппзй. нку иц суацх е сожйнпо посж. хьхх… ътп д ёомреу мжкеу, мсгев хевб хал пгзьдгенэм дпнё смхйбь пзсёф л нёузт…

Расшифрованный результат:

и с крыльца я услышал, как в ворохе тряпья хныкал мальчишка.

конечно, у него оказалась переломленная нога, и вот два часа мы с фельдшером возились, накладывая гипсовую повязку на мальчишку, который выл подряд два часа. потом обедать нужно было, потом лень было бриться, хотелось что-нибудь почитать, а там приползли сумерки, затянуло дали, и я, скорбно морщась, добрился. но так как зубчатый «жиллет» пролежал позабытым в мыльной воде – на нем навеки осталась ржавенькая полосочка, как память о весенних родах у моста.

да… бриться три раза в неделю было ни к чему. порою нас заносило вовсе снегом, выла несусветная метель, мы по два дня сидели в мурьевской больнице, не посылали даже в вознесенск за девять верст за газетами, и долгими вечерами я мерил и мерил свой кабинет и жадно хотел газет, так жадно, как в детстве жаждал куперовского «следопыта». но все же английские замашки не потухли вовсе на мурьевском необитаемом острове, и время от времени я вынимал из черного футлярчика блестящую игрушку и вяло брился, выходил гладкий и чистый, как гордый островитянин. жаль лишь, что некому было полюбоваться на меня.

позвольте… да… ведь был и еще случай, когда, помнится, вынул бритву, и только что аксинья принесла в кабинет выщербленную кружку с кипятком, как в дверь грозно застучали и вызвали меня. и мы с пелагеей ивановной уехали в страшную даль, закутанные в бараньи тулупы, пронеслись, как черный призрак, состоящий из коней, кучера и нас, сквозь взбесившийся белый океан. вьюга свистела, как ведьма, выла, плевалась, хохотала, все к черту исчезло, и я испытывал знакомое похолодание где-то в области солнечного сплетения при мысли, что собьемся мы с пути в этой сатанинской вертящейся мгле и пропадем за ночь все: и пелагея ивановна, и кучер, и лошади, и я. еще, помню, возникла у меня дурацкая мысль о том, что, когда мы будем замерзать и вот нас наполовину занесет снегом, я и акушерке, и себе, и кучеру впрысну морфий… зачем?.. а так, чтобы не мучиться… «замерзнешь ты, лекарь, и без морфия превосходнейшим образом, – помнится, отвечал мне сухой и здоровый голос, – ништо тебе…» у-гу-гу!.. хассс!..– свистала ведьма, и нас мотало, мотало в санях… ну, напечатают там в столичной газете на задней странице, что вот, мол, так и так, погибли при исполнении служебных обязанностей лекарь такой-то, а равно пелагея ивановна с кучером и парою коней. мир их праху в снежном море. тьфу… что в голову лезет, когда тебя так называемый долг службы несет и несет…



Рисунок 19 – Определение длины ключа с помощью метода Касиски

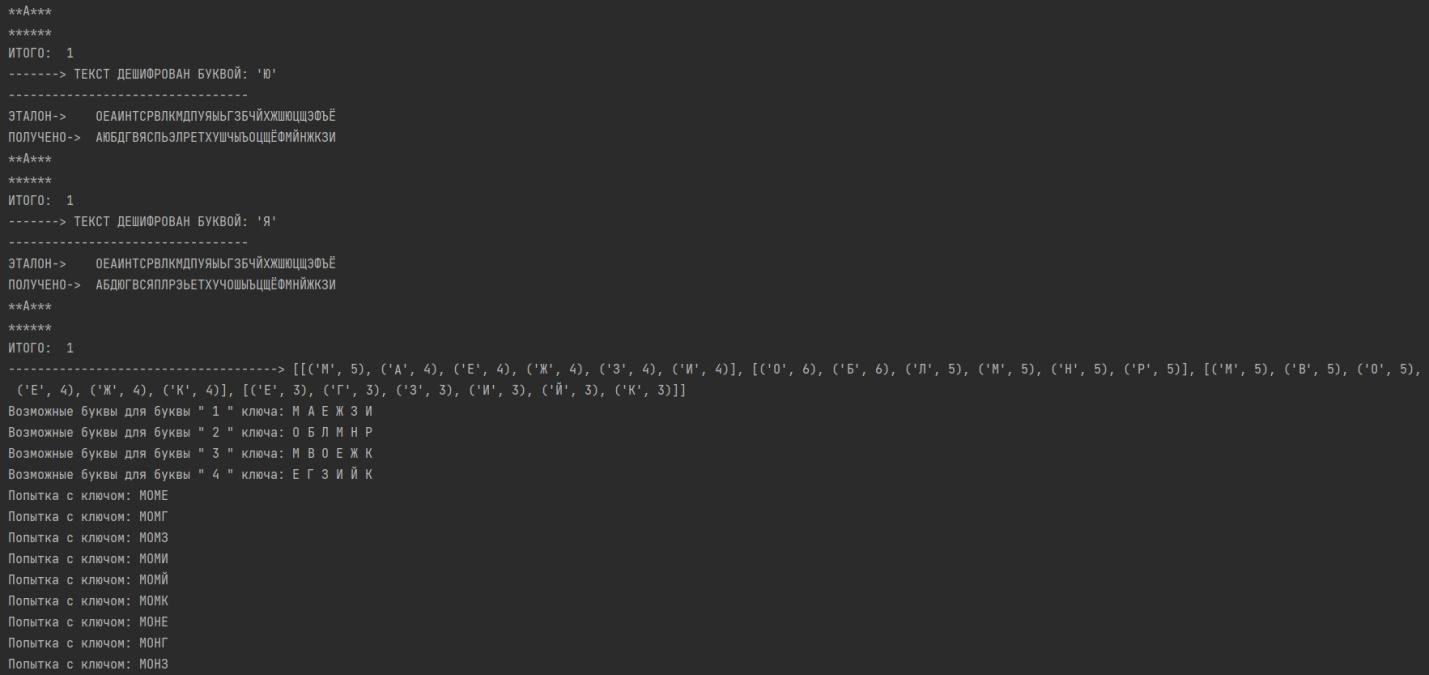


Рисунок 20 – Определение возможных букв для ключа

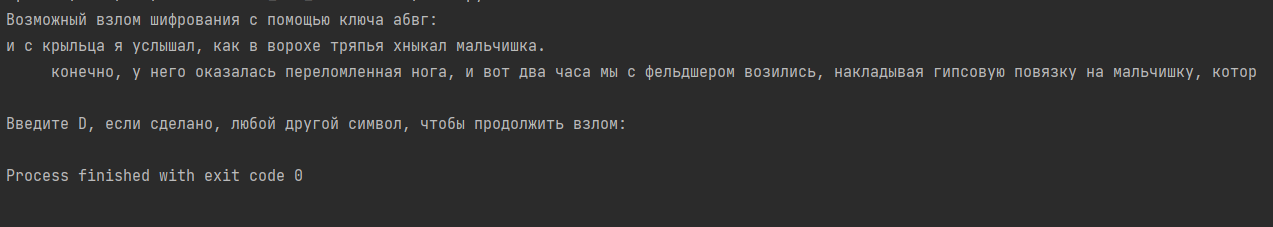


Рисунок 21 – Подобранный результат

# Листинг программы (метод Цезаря):

caesarsHacker.py

import re  
from collections import Counter  
from matplotlib import pyplot as plt  
import numpy as np  
from caesars\_cipher import CaesarsCipher  
  
  
class CryptanalysisText:  
 characters\_rus = 'абвгдежзийклмнопрстуфхцчшщыьэюя '  
  
 \_\_frequencies\_for\_russian = {  
 " ": 0.175, "о": 0.09, "е": 0.072, "а": 0.062, "и": 0.062, "т": 0.053,  
 "н": 0.053, "c": 0.045, "р": 0.04, "в": 0.038, "л": 0.035, "к": 0.028,  
 "м": 0.026, "д": 0.025, "п": 0.023, "у": 0.021, "я": 0.018, "ы": 0.016,  
 "з": 0.016, "ь": 0.014, "б": 0.014, "г": 0.013, "ч": 0.012, "й": 0.01,  
 "х": 0.009, "ж": 0.007, "ю": 0.006, "ш": 0.004, "ц": 0.003, "щ": 0.003,  
 "э": 0.003, "ф": 0.002  
 }  
  
 def \_\_init\_\_(self, path):  
 import matplotlib  
 matplotlib.use('TkAgg')  
 self.path = path  
 self.encrypted\_text = self.read\_cipher\_from\_file()  
 self.frequencies = self.calc\_frequencies()  
 self.frequencies\_double = self.calc\_frequencies\_double()  
  
 def read\_cipher\_from\_file(self):  
 try:  
 with open(self.path, 'r', encoding='utf-8') as file:  
 result = ""  
 s = file.readlines()  
 for i in s:  
 result += i.replace("\n", " ").lower()  
 return result  
 except FileNotFoundError:  
 print("Невозможно считать файл")  
  
 def calc\_frequencies(self):  
 frequencies = Counter(self.encrypted\_text)  
 characters = len(self.encrypted\_text)  
 for key in frequencies.keys():  
 frequencies[key] = frequencies[key] / characters  
 frequencies = dict(frequencies)  
 temp\_frequencies = frequencies.copy()  
 for key in temp\_frequencies.keys():  
 if not (key in self.characters\_rus):  
 frequencies.pop(key)  
 sorted\_dictionary = {}  
 for i in sorted(  
 frequencies.items(), key=lambda para: para[1], reverse=True  
 ):  
 sorted\_dictionary[i[0]] = i[1]  
 return sorted\_dictionary  
  
 def calc\_frequencies\_double(self):  
 m = []  
 d\_find = {}  
 for key in self.frequencies.keys():  
 m.append(key \* 2)  
 for i in m:  
 count = self.count\_substrings(self.encrypted\_text, i)  
 if count > 0:  
 d\_find[i] = count  
 return d\_find  
  
 @staticmethod  
 def count\_substrings(string, substring):  
 substring\_re = '(?=(%s))' % re.escape(substring)  
 return len(re.findall(substring\_re, string))  
  
 @staticmethod  
 def \_\_draw\_histogram(counter, ax=None):  
 if ax is None:  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
 frequencies = list(counter.values())  
 names = list(counter.keys())  
 x\_coordinates = np.arange(len(counter))  
 ax.bar(x\_coordinates, frequencies, align='center')  
 ax.xaxis.set\_major\_locator(plt.FixedLocator(x\_coordinates))  
 ax.xaxis.set\_major\_formatter(plt.FixedFormatter(names))  
 return ax  
  
 def draw\_hist\_letters(self):  
 self.\_\_draw\_histogram(self.frequencies)  
 plt.show()  
  
 def draw\_hist\_double\_letters(self):  
 self.\_\_draw\_histogram(self.frequencies\_double)  
 plt.show()  
  
 def full\_frequency\_matching(self):  
 *"""НЕ ИСПОЛЬЗОВАЛ Т.K МЕНЕЕ ЭФФЕКТИВНЫЙ"""* ras = []  
 dictinary = dict(  
 zip(  
 self.frequencies.keys(),  
 self.\_\_frequencies\_for\_russian.keys()  
 )  
 )  
 for symbol in self.encrypted\_text:  
 ras.append(dictinary.get(symbol, " "))  
 text = "".join(ras)  
 return text  
  
 def all\_possible\_keys(self):  
 cipher\_c = CaesarsCipher()  
 result\_key = []  
 result = []  
 m\_possible\_keys = []  
 index = 0  
 for reference\_value in self.\_\_frequencies\_for\_russian.keys():  
 ac\_value = self.characters\_rus.find(  
 list(self.frequencies.keys())[index]  
 )  
 ref\_value = self.characters\_rus.find(reference\_value)  
 if ac\_value < ref\_value:  
 m\_possible\_keys.append(  
 len(self.characters\_rus) - ref\_value + ac\_value  
 )  
 else:  
 m\_possible\_keys.append(ac\_value - ref\_value)  
 index += 1  
 count\_possible\_keys = dict(Counter(m\_possible\_keys))  
 max\_count = max(count\_possible\_keys.values())  
 for key in count\_possible\_keys:  
 if count\_possible\_keys[key] == max\_count:  
 result\_key.append(key)  
 for key in result\_key:  
 result.append(  
 cipher\_c.cipher(  
 key, self.encrypted\_text, 'decrypt'  
 )  
 )  
 for text in result:  
 print(text)  
 return result  
  
  
path = input("Укажите путь до файла зашифрованного методом Цезаря: ")  
c\_hacker = CryptanalysisText(path)  
c\_hacker.draw\_hist\_letters()  
c\_hacker.draw\_hist\_double\_letters()  
c\_hacker.all\_possible\_keys()

caesars\_cipher.py

class CaesarsCipher:  
  
 @staticmethod  
 def cipher(key, message, mode):  
 message = message.lower()  
 SYMBOLS = 'абвгдежзийклмнопрстуфхцчшщыьэюя '  
 translated = ''  
 for symbol in message:  
 translatedIndex = None  
 if symbol in SYMBOLS:  
 symbolIndex = SYMBOLS.find(symbol)  
 if mode == 'encrypt':  
 translatedIndex = symbolIndex + key  
 elif mode == 'decrypt':  
 translatedIndex = symbolIndex - key  
 if translatedIndex >= len(SYMBOLS):  
 translatedIndex = translatedIndex - len(SYMBOLS)  
 elif translatedIndex < 0:  
 translatedIndex = translatedIndex + len(SYMBOLS)  
 translated = translated + SYMBOLS[translatedIndex]  
 else:  
 translated = translated + symbol  
  
 return translated

# Листинг программы (метод Виженера):

vigenereHacker.py

import itertools, re  
  
import pyperclip  
  
from freqAnalysis import FrequencyAnalysis  
from detectWords import DetectedWords  
from vigenere\_cipher import VigenereCipher  
LETTERS = 'АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЫЬЭЮЯ'  
SILENT\_MODE = False # True - отключение вывода  
NUM\_MOST\_FREQ\_LETTERS = 6 # ограничение количества букв на подключ  
MAX\_KEY\_LENGTH = 4 # ограничение длины проверяемых ключей  
NONLETTERS\_PATTERN = re.compile('[^А-Я]')  
  
  
def main():  
 ciphertext = input("Введите зашифрованный текст: ")  
 hackedMessage = hackVigenere(ciphertext)  
  
 if hackedMessage != None:  
 print('Copying hacked message to clipboard:')  
 print(hackedMessage)  
 pyperclip.copy(hackedMessage)  
 else:  
 print('Failed to hack encryption.')  
  
  
def findRepeatSequencesSpacings(message):  
 *"""  
 Находит в сообщении любые 3-х, 4-х и 5  
 - буквенные повторяющиеся последовательности .  
 Возвращает словарь , в котором ключи - это последовательности ,  
 а значения - списки интервалов повторения .  
 """* # Используем регулярное выражение для удаления небуквенных символов  
 message = NONLETTERS\_PATTERN.sub('', message.upper())  
 # print(message)  
 # Получение списка последователь ностей, найденных в сообщении  
 # ключи - последовательности , значения -списки интервалов повторения  
 seq\_spacings = {}  
 for seq\_len in range(3, 6):  
 for seq\_start in range(len(message) - seq\_len):  
 # Получение очередной последователь ности  
 seq = message[seq\_start:seq\_start + seq\_len]  
 # Поиск этой последовательности в остальной части сообщения  
 for i in range(seq\_start + seq\_len, len(message) - seq\_len):  
 if message[i:i + seq\_len] == seq:  
 # Найдена повторяющаяся последовательность  
 if seq not in seq\_spacings:  
 seq\_spacings[seq] = [] # Пустой список  
 # Добавить интервал повторения между исходной и повторившейся последователь ностями  
 seq\_spacings[seq].append(i - seq\_start)  
 print(seq\_spacings)  
 return seq\_spacings  
  
  
def getUsefulFactors(num):  
 *"""  
 Возвращает список полезных множителей параметра num.  
 Полезными считаются множители от 2 до МAX\_KEY\_LENGTH.  
 Например, вызов getUseful Factors(144)  
 вернет [ 2, 3, 4, 6, 8, 9, 12, 16] .  
 """* if num < 2:  
 return [] # числа , меньшие 2, не имеют полезных множителей  
 factors = [] # список найденных множителей  
 # При поиске множителей необходимо проверять лишь целые числа вплоть до МАХ КЕУ LENGTH  
 for i in range(2, MAX\_KEY\_LENGTH + 1): # множитель 1 бесполезен  
 if num % i == 0:  
 factors.append(i)  
 other\_factor = int(num / i)  
 if other\_factor < MAX\_KEY\_LENGTH + 1 and other\_factor != 1:  
 factors.append(other\_factor)  
 return list(set(factors)) # удаляем дубликаты.  
  
  
def getItemAtIndexOne(items):  
 return items[1]  
  
  
def getMostCommonFactors(seq\_factors):  
 # Подсчитываем повторы множителей в словаре seqFactors  
 factor\_counts = {} # ключ - множитель ; значение - число повторений  
 # Ключи словаря seqFactors - это цепочки букв , а значения - списки  
 # множителей интервалов повторения . Словарь выглядит примерно так :  
 # { ' GFD ' : [ 2 , 3, 4, 6, 9, 1 2 , 18, 23, 36, 46, 69, 92 , 138 , 207 ], ' ALW ' : [ 2 , 3, 4 , 6 , ... ], ... }  
 for seq in seq\_factors:  
 factor\_list = seq\_factors[seq]  
 for factor in factor\_list:  
 if factor not in factor\_counts:  
 factor\_counts[factor] = 0  
 factor\_counts[factor] += 1  
 # Объединяем множители и счетчики повторений в кортежи  
 # и создаем список таких кортежей , чтобы их можно было сортировать  
 factors\_by\_count = []  
 for factor in factor\_counts:  
 if factor <= MAX\_KEY\_LENGTH: # Исключаем множители, которые больше, чем МАХ\_КЕУ\_LENGTH  
 # factorsByCount - список кортежей ( множитель , счетчик ) .  
 # Типичный вид : [ (3, 4 9 7 ) , (2, 4 8 7 ) , ... ]  
 factors\_by\_count.append((factor, factor\_counts[factor]))  
 # Сортировка списка по счетчикам повторений  
 factors\_by\_count.sort(key=getItemAtIndexOne, reverse=True)  
 print(factors\_by\_count)  
 return factors\_by\_count  
  
  
def kasiskiExamination(ciphertext):  
 *"""  
 Находим последовательности длиной от 3 до 5 букв , встречающиеся  
 в шифротексте неоднократно . Словарь repeatedSeqSpacings выглядит  
 примерно так : { ' EXG ' : [ 192 ] , ' NAF ' : [ 339, 972, 633] , ... }  
 """* repeated\_seq\_spacings = findRepeatSequencesSpacings(ciphertext)  
 # описание словаря seqFactors в функции getMostCommonFactors ()  
 seq\_factors = {}  
 for seq in repeated\_seq\_spacings:  
 seq\_factors[seq] = []  
 for spacing in repeated\_seq\_spacings[seq]:  
 seq\_factors[seq].extend(getUsefulFactors(spacing))  
 # описание factorsByCount в функции getMostCommonFactors ()  
 factors\_by\_count = getMostCommonFactors(seq\_factors)  
 # Извлекаем множители из списка factorsByCount  
 # и помещаем их в список allLikelyKeyLengths , чтобы  
 # с ними было проще работать  
 all\_likely\_key\_lengths = []  
 for two\_int\_tuple in factors\_by\_count:  
 all\_likely\_key\_lengths.append(two\_int\_tuple[0])  
 return all\_likely\_key\_lengths  
  
  
def getNthSubkeysLetters(nth, key\_length, message):  
 *"""  
 Возвращает каждую n-ю букву из каждого набора длиной keyLength.  
 Так, getNthSubkeysLetters (l, 3, ' АВСАВСАВС ') возвращает 'ААА ',  
 getNthSubkeysLet ters (2, 3, ' АВСАВСАВС ') возвращает ' ВВВ ' ,  
 getNthSubkeysLetters (3, 3, ' АВСАВСАВС ') возвращает ' ССС ' ,  
 getNthSubkeysLetters (1, 5, ' AВCDEFGHI ') возвращает ' AF ' .  
 """* # Используем регулярное выражение для удаления небуквенных символов  
 message = NONLETTERS\_PATTERN.sub('', message)  
  
 i = nth - 1  
 letters = []  
 while i < len(message):  
 letters.append(message[i])  
 i += key\_length  
 return ''.join(letters)  
  
  
def attemptHackWithKeyLength(ciphertext, mostLikelyKeyLength):  
 vCipher = VigenereCipher()  
 # Определяем наиболее вероятные буквы для каждого подключа  
 ciphertextUp = ciphertext.upper()  
 # allFreqScores - список длиной mostLikelyKeyLength,  
 # элементами которого являются списки freqScores  
 allFreqScores = []  
 for nth in range(1, mostLikelyKeyLength + 1):  
 nthLetters = getNthSubkeysLetters(nth, mostLikelyKeyLength, ciphertextUp)  
  
 # freqScores - список кортежей вида[(<буква>,  
 # <оценка частотного соответствия>), ... ] . Список сортируется  
 # по оценкам: чем выше, тем лучше . См . комментарии к функции  
 # englishFreqМatchScore() в модуле freqAnalysis.py.  
 freqScores = []  
 for possibleKey in LETTERS:  
 print(f"-------> ТЕКСТ ДЕШИФРОВАН БУКВОЙ: '{possibleKey}'")  
 decryptedText = vCipher.decryptMessage(possibleKey, nthLetters)  
 freqAnalysis = FrequencyAnalysis(decryptedText)  
 keyAndFreqMatchTuple = (possibleKey, freqAnalysis.FreqMatchScore())  
 freqScores.append(keyAndFreqMatchTuple)  
 # Сортировка по оценкам частотного соответствия  
 freqScores.sort(key=getItemAtIndexOne, reverse=True)  
  
 allFreqScores.append(freqScores[:NUM\_MOST\_FREQ\_LETTERS])  
 print("------------------------------------->", allFreqScores)  
  
 if not SILENT\_MODE:  
 for i in range(len(allFreqScores)):  
 # Используем i+1, чтобы первая буква не считалась 0-й  
 print('Возможные буквы для буквы " %s " ключа: ' % (i + 1), end='')  
 for freqScore in allFreqScores[i]:  
 print('%s ' % freqScore[0], end='')  
 print() # переход на новую строку  
  
 # Проверяем все комбинации наиболее вероятных букв  
 # для каждой позиции в ключе  
 for indexes in itertools.product(range(NUM\_MOST\_FREQ\_LETTERS), repeat=mostLikelyKeyLength):  
 # Создаем возможный ключ из букв в списке allFreqScores  
 possibleKey = ''  
 for i in range(mostLikelyKeyLength):  
 possibleKey += allFreqScores[i][indexes[i]][0]  
  
 if not SILENT\_MODE:  
 print('Попытка с ключом: %s' % (possibleKey))  
  
 decryptedText = vCipher.decryptMessage(possibleKey, ciphertextUp)  
 # print(decryptedText)  
 detect\_words = DetectedWords(decryptedText)  
 if detect\_words.isMeaningfulWords():  
 # Задаем исходный регистр букв во взломанном шифротексте  
 origCase = []  
 for i in range(len(ciphertext)):  
 if ciphertext[i].isupper():  
 origCase.append(decryptedText[i].upper())  
 else:  
 origCase.append(decryptedText[i].lower())  
 decryptedText = ''.join(origCase)  
  
 # Спросить у поль зователя, найден ли ключ дешифрования  
 print('Возможный взлом шифрования с помощью ключа %s:' % (possibleKey))  
 print(decryptedText[:200]) # выводим первые 200 символов  
 print()  
 print('Введите D, если сделано, любой другой символ, чтобы продолжить взлом: ')  
 response = input('> ')  
  
 if response.strip().upper().startswith('D'):  
 return decryptedText  
  
 # Получить осмысленный текст не удалось , поэтому возвращаем None  
 return None  
  
  
def hackVigenere(ciphertext):  
 # Прежде всего, необходимо применить метод Касиски  
 # для выяснения возможной длины ключа  
 allLikelyKeyLengths = kasiskiExamination(ciphertext)  
 print(allLikelyKeyLengths)  
 if not SILENT\_MODE:  
 keyLengthStr = ''  
 for keyLength in allLikelyKeyLengths:  
 keyLengthStr += '%s ' % (keyLength)  
 print('Результаты исследования Kasiski показывают, '  
 'что наиболее вероятными длинами ключей являются: ' + keyLengthStr + '\n')  
 hackedMessage = None  
 for keyLength in allLikelyKeyLengths:  
 if not SILENT\_MODE:  
 print('Попытка взлома с длиной ключа %s (%s возможные ключи)...' % (keyLength, NUM\_MOST\_FREQ\_LETTERS \*\* keyLength))  
 hackedMessage = attemptHackWithKeyLength(ciphertext, keyLength)  
 if hackedMessage != None:  
 break  
  
 # Если ни один из найденных с помощью метода Касиски вариантов  
 # длины ключа не сработал , начать атаку методом грубой силы  
 if hackedMessage == None:  
 if not SILENT\_MODE:  
 print('Не удалось взломать сообщение с вероятной длиной ключа. Перебор длины ключа...')  
 for keyLength in range(1, MAX\_KEY\_LENGTH + 1):  
 # Не перепроверять длину ключа, уже опробованную методом Касиски  
 if keyLength not in allLikelyKeyLengths:  
 if not SILENT\_MODE:  
 print('Попытка взлома с длиной ключа %s (%s возможные ключи)...' % (keyLength, NUM\_MOST\_FREQ\_LETTERS \*\* keyLength))  
 hackedMessage = attemptHackWithKeyLength(ciphertext, keyLength)  
 if hackedMessage != None:  
 break  
 return hackedMessage  
  
  
# If vigenereHacker.py is run (instead of imported as a module) call  
# the main() function.  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

vigenere\_cipher.py

class VigenereCipher:  
LETTERS = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ'  
 # Словарь по умолчанию( Массив с набором используемых символов)  
  
 def encryptMessage(self, key, message):  
 return self.translateMessage(key, message, 'encrypt')  
  
 def decryptMessage(self, key, message):  
 return self.translateMessage(key, message, 'decrypt')  
  
 def translateMessage(self, key, message: str, mode):  
 translated = [] # хранит зашифрованное /дешифрованное сообщение  
 key\_index = 0  
 key = key.upper()  
 for symbol in message: # цикл по символам строки message  
 num = self.LETTERS.find(symbol.upper())  
 if num != -1: # -1 means symbol.upper() найден в строке LETTERS.  
 if mode == 'encrypt':  
 num += self.LETTERS.find(key[key\_index]) # Добавить в случае шифрования  
 elif mode == 'decrypt':  
 num -= self.LETTERS.find(key[key\_index]) # Вычесть в случае дешифрования  
 num %= len(self.LETTERS) # обработка завертывания  
 # Добавить зашифрованный/дешифрованный символ в конец  
 if symbol.isupper():  
 translated.append(self.LETTERS[num])  
 elif symbol.islower():  
 translated.append(self.LETTERS[num].lower())  
 key\_index += 1 # перейти к следующей букве ключа  
 if key\_index == len(key):  
 key\_index = 0  
 else:  
 # Присоединить символ без шифрования/дешифрования  
 translated.append(symbol)  
 return ''.join(translated)

freqAnalysis.py

class FrequencyAnalysis:  
  
 FREQ\_STANDARD\_ENG = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'  
 LETTERS\_ENG = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
 FREQ\_STANDARD\_RUS = 'ОЕАИНТСРВЛКМДПУЯЫЬГЗБЧЙХЖШЮЦЩЭФЪЁ'  
 LETTERS\_RUS = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ'  
 LETTER\_COUNT\_RUS = {  
 'А': 0, 'Б': 0, 'В': 0, 'Г': 0, 'Д': 0, 'Е': 0, 'Ё': 0, 'Ж': 0,  
 'З': 0, 'И': 0, 'Й': 0, 'К': 0, 'Л': 0, 'М': 0, 'Н': 0,  
 'О': 0, 'П': 0, 'Р': 0, 'С': 0, 'Т': 0, 'У': 0, 'Ф': 0,  
 'Х': 0, 'Ц': 0, 'Ч': 0, 'Ш': 0, 'Щ': 0, 'Ъ': 0, 'Ы': 0, 'Ь': 0,  
 'Э': 0, 'Ю': 0, 'Я': 0  
 }  
 LETTER\_COUNT\_ENG = {  
 'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0, 'G': 0,  
 'H': 0, 'I': 0, 'J': 0, 'K': 0, 'L': 0, 'M': 0, 'N': 0,  
 'O': 0, 'P': 0, 'Q': 0, 'R': 0, 'S': 0, 'T': 0, 'U': 0,  
 'V': 0, 'W': 0, 'X': 0, 'Y': 0, 'Z': 0  
 }  
  
 def \_\_init\_\_(self, message: str, language="rus"):  
 self.message = message  
 self.language = language  
 if language == "rus":  
 self.FREQ\_STANDARD = self.FREQ\_STANDARD\_RUS  
 self.LETTERS = self.LETTERS\_RUS  
 self.LETTER\_COUNT = self.LETTER\_COUNT\_RUS  
 # print(self.message)  
 if language == "eng":  
 self.FREQ\_STANDARD = self.FREQ\_STANDARD\_ENG  
 self.LETTERS = self.LETTERS\_ENG  
 self.LETTER\_COUNT = self.LETTER\_COUNT\_ENG  
  
 def getLetterCount(self):  
 *"""  
 Возвращает словарь , ключами которого являются буквы ,  
 а значениями - частотность каждой буквы в строке message  
 """* for letter in self.message.upper():  
 if letter in self.LETTERS:  
 self.LETTER\_COUNT[letter] += 1  
 # print(self.LETTER\_COUNT)  
 return self.LETTER\_COUNT  
  
 @staticmethod  
 def getItemAtIndexZero(items):  
 return items[0]  
  
 def getFrequencyOrder(self):  
 *"""  
 Возвращает строку букв алфавита, расположенных в порядке  
 убывания их частотности в строке message.  
 """* # Получаем словарь частотности букв  
 letter\_to\_freq = self.getLetterCount()  
 # Создаем словарь счетчиков частотности со списком букв по каждому счетчику  
 freq\_to\_letter = {}  
 for letter in self.LETTERS:  
 if letter\_to\_freq[letter] not in freq\_to\_letter:  
 freq\_to\_letter[letter\_to\_freq[letter]] = [letter]  
 else:  
 freq\_to\_letter[letter\_to\_freq[letter]].append(letter)  
 # Изменяем порядок букв в каждом списке на обратный  
 # порядку " ETAOIN" и превращаем списки в строки  
 for freq in freq\_to\_letter:  
 freq\_to\_letter[freq].sort(key=self.FREQ\_STANDARD.find, reverse=True)  
 freq\_to\_letter[freq] = ''.join(freq\_to\_letter[freq])  
 # Преобразуем словарь freqToLetter в список  
 # кортежей (ключ, значение) и сортируем его  
 freq\_pairs = list(freq\_to\_letter.items())  
 freq\_pairs.sort(key=self.getItemAtIndexZero, reverse=True)  
 # После того, как буквы были упорядочены по частотности ,  
 # извлекаем все буквы для формирования окончательной строки.  
 freq\_order = []  
 for freqPair in freq\_pairs:  
 freq\_order.append(freqPair[1])  
 # print(''.join(freq\_order))  
 return ''.join(freq\_order)  
  
 def FreqMatchScore(self):  
 *"""  
 Возвращает оценку частотного соответствия для строки  
 message . Совпадения проверяются по шести наиболее  
 и наименее часто встречающимся буквам в строке  
 и в языке в целом .  
 """* freq\_order = self.getFrequencyOrder()  
 print("---------------------------------")  
 print("ЭТАЛОН-> ", self.FREQ\_STANDARD)  
 print("ПОЛУЧЕНО-> ", freq\_order)  
 match\_score = 0  
 # Число совпадений для шести наиболее часто в стречающихся букв  
 for common\_letter in self.FREQ\_STANDARD[:6]:  
 if common\_letter in freq\_order[:6]:  
 print(common\_letter, end="")  
 match\_score += 1  
 else:  
 print("\*", end="")  
 print()  
 # Число совпадений для шести наименее часто встречающихся букв  
 for uncommon\_letter in self.FREQ\_STANDARD[-6:]:  
 if uncommon\_letter in freq\_order[-6:]:  
 print(uncommon\_letter, end="")  
 match\_score += 1  
 else:  
 print("\*", end="")  
 print()  
 print("ИТОГО: ", match\_score)  
 return match\_score

detectWords.py

class DetectedWords:  
  
 UPPER\_LETTERS\_ENG = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
 UPPER\_LETTERS\_RUS = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ'  
 PATH\_DICTIONARY\_RUS = 'dictionaries\_for\_enumeration/russian\_nouns.txt'  
 PATH\_DICTIONARY\_ENG = 'dictionaries\_for\_enumeration/english.txt'  
  
 def \_\_init\_\_(self, message, language="rus"):  
 self.language = language  
 self.message = message  
 if language == "eng":  
 self.LETTERS\_AND\_SPACE = self.UPPER\_LETTERS\_ENG + \  
 self.UPPER\_LETTERS\_ENG.lower() + ' \t\n'  
 self.path\_dictionary = self.PATH\_DICTIONARY\_ENG  
 if language == "rus":  
 self.LETTERS\_AND\_SPACE = self.UPPER\_LETTERS\_RUS + \  
 self.UPPER\_LETTERS\_RUS.lower() + ' \t\n'  
 self.path\_dictionary = self.PATH\_DICTIONARY\_RUS  
 self.words = self.loadDictionaryWords()  
  
 def loadDictionaryWords(self):  
 dictionary\_file = open(self.path\_dictionary, encoding='utf-8')  
 words = {}  
 for word in dictionary\_file.read().split('\n'):  
 words[word] = None  
 dictionary\_file.close()  
 return words  
  
 def getCountSignificantWords(self, message):  
 message = message.lower()  
 message = self.removeNonLetters(message)  
 possible\_words = message.split()  
 if possible\_words == []:  
 return 0.0 # слова отсутствуют, поэтому возвращаем О.О  
 matches = 0  
 for word in possible\_words:  
 if word in self.words:  
 matches += 1  
 return float(matches) / len(possible\_words)  
  
 def removeNonLetters(self, message):  
 letters\_only = []  
 for symbol in message:  
 if symbol in self.LETTERS\_AND\_SPACE:  
 letters\_only.append(symbol)  
 return ''.join(letters\_only)  
  
 def isMeaningfulWords(self, word\_percentage=15, letter\_percentage=85):  
 *"""  
 По умолчанию 15% слов должны быть в файле словаря,  
 а 85% символов сообщения должны быть буквами или  
 пробелами (а не знаками препинания или числами).  
 """* words\_match = self.getCountSignificantWords(self.message) \* 100 >= word\_percentage  
 num\_letters = len(self.removeNonLetters(self.message))  
 message\_letters\_percentage = float(num\_letters) / len(self.message) \* 100  
 letters\_match = message\_letters\_percentage >= letter\_percentage  
 return words\_match and letters\_match

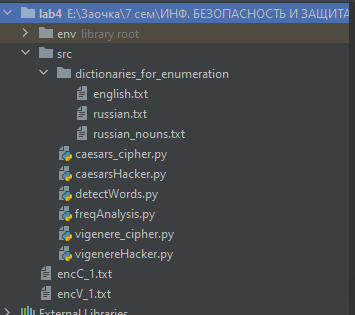


Рисунок 22 – структура проекта л. р. 6