Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Прикладная математика и компьютерная безопасность

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 14**

по Криптографические методы защиты информации

наименование дисциплины

Криптоанализ полиалфавитных шифров

тема

Преподаватель В.И. Вайнштейн

подпись**,** дата инициалы, фамилия

Студент КИ17-01, 031722011 К.А. Василенко

номер группы, зачетной книжки подпись**,** дата инициалы, фамилия

Красноярск 2021

# ВВЕДЕНИЕ

Цель работы: получение практических навыков в криптоанализе полиалфавитных шифров, реализовать анализ полиалфавитного шифротекста на любом языке программирования. Предусмотреть графический интерфейс.

Задачи:

1. реализовать программно анализ полиалфавитного шифротекста, предусматривая графический интерфейс;
2. провести тесты на работоспособность программы;
3. сделать отчёт о проделанной работе.

**Реализация шифра**

1. **Описание метода**

**Индекс совпадений:**

Индекс совпадений — один из методов криптоанализа шифра Виженера. Метод основывается на вычислении вероятности того, что два случайных элемента текста совпадут. Эту вероятность называют индексом совпадений.

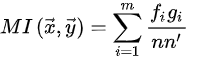
Рассмотрим текст, написанный на некотором языке. Алфавит данного языка будем полагать состоящим из m{\displaystyle m} символов. Рассмотрим достаточно длинную строку x->{\displaystyle {\vec {x}}} из n{\displaystyle n} символов. Индексом совпадения называют вероятность совпадения двух произвольных символов в строке. Если fi{\displaystyle f\_{i}} — количество i{\displaystyle i}-го символа алфавита в строке{\displaystyle {\vec {x}}}, то индекс совпадения вычисляется по формуле:



Ic для русского языка – 0.0553

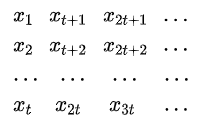
Ic для английского языка – 0.0644

Рассмотрим две строки x->{\displaystyle {\vec {x}}} и y->{\displaystyle {\vec {y}}} с длинами n{\displaystyle n} и n’ {\displaystyle n'}соответственно. Алфавит, как и прежде, состоит из m{\displaystyle m} символов. Взаимным индексом совпадений этих строк называют вероятность того, что символ, случайно выбранный из первой строки, совпадёт со случайно выбранным символом второй строки. Пусть fi, gi{\displaystyle f\_{i},g\_{i}} — количество i{\displaystyle i}-го символа алфавита в первой и второй строках соответственно. Тогда взаимный индекс совпадений будет равен:



Алгоритм нахождения длины ключа

Разобьём текст x1 x2 … xn на столбцы размера t.



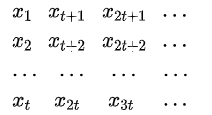
Если {\displaystyle t}t кратно длине ключа, то каждые два элемента текста, отстоящие друг от друга на {\displaystyle a\times t}a x t позиций, {\displaystyle a\in N}a € N, зашифрованы одним и тем же алфавитом. А это означает, что каждая строка в выписанной выше таблице получена из открытого текста перестановкой. Если же {\displaystyle t}t не кратно длине ключа, то строки являются полиалфавитным шифром.

Индекс совпадений для перестановки открытого текста и для полиалфавитного шифра заметно отличается. Таким образом, перебирая различные значения {\displaystyle t}t и вычисляя для каждого из них индекс совпадений, мы можем выделить те {\displaystyle t}t, которые кратны длине ключа. Определить длину ключа по этим данным не составляет труда.

Алгоритм нахождения ключа

Предположим, мы определили длину ключа {\displaystyle t}t. Найдём теперь сам ключ.

Вновь выпишем текст в столбцы размера {\displaystyle t}t.



Рассмотрим две строки этой таблицы. Сдвинем алфавит одной из строк на {\displaystyle s}s символов и вычислим взаимный индекс совпадений полученных строк. Т.к. каждая из этих двух строк получена сдвигом алфавита открытого текста, то максимум взаимного индекса совпадений будет наблюдаться при нулевом конечном относительном сдвиге.

Поэтому применяется следующий алгоритм: вычисляется взаимный индекс совпадений для различных {\displaystyle s}s, ищется значение {\displaystyle s}s, при котором взаимный индекс совпадений максимален. Тогда начальный относительный сдвиг строк будет равен {\displaystyle m-s}m - s ({\displaystyle m}m — размер алфавита). Вычисляются относительные сдвиги между всеми парами строк. Т.к. сдвиги строк таблицы соответствуют сдвигам букв ключа, то остаётся перебрать {\displaystyle m}m возможных ключей и выбрать из них наиболее правдоподобный.

**Автокорреляционный метод:**

Автокорелляционный метод позволяет криптоаналитику найти длину {\displaystyle l}l ключевого слова, используемого в полиалфавитном шифре. Как только длина ключевого слова обнаружена, криптоаналитик выстраивает зашифрованный текст в {\displaystyle l}l колонках. При этом каждую колонку можно рассматривать как зашифрованный моноалфавитным шифром текст, который можно подвергнуть частотному анализу.

Сам метод состоит в том, что исходный шифротекст {\displaystyle (c\_{1},c\_{2},\dotsc ,c\_{L})}(c1, c2, …, cL) выписывается в строку, а под ней выписываются строки, полученные сдвигом вправо на {\displaystyle t=1,2,3,\dotsc }t = 1, 2, 3, … позиций. Для каждого {\displaystyle t}t подсчитывается число {\displaystyle n\_{t}}nt совпадений {\displaystyle c\_{i}=c\_{i+t}}ci = ci+t, где {\displaystyle i\in \left[1,L-t\right]}i € [1, L – t] и вычисляются автокорреляционные коэффициенты **γ**{\displaystyle \gamma \_{t}}:

**γt =**

Для сдвигов, кратных периоду, коэффициенты **γt**{\displaystyle \gamma \_{t}} должны быть заметно больше, чем для сдвигов, не кратных периоду, и иметь значение близкое к индексу совпадений используемого языка (для русского языка ~ 0.0553). Это объясняется следующим образом. Когда величина сдвига кратна длине ключевого слова, символы {\displaystyle c\_{i}}ci и {\displaystyle c\_{i+t}}ci + t шифруются одинаковым моноалфавитным шифром, что не изменяет факт их совпадения. А так как индекс совпадений вводится как вероятность совпадения двух произвольных букв в строке, то для сдвигов, кратных или равных периоду, автокорреляционные коэффициенты, при достаточно большой длине текста, будут близки к индексу совпадений естественного языка.{\displaystyle \gamma \_{t}={\dfrac {n\_{t}}{L-t}}}

**Метод Касиски:**

Если повторяющаяся подстрока в открытом тексте зашифровывается одной и той же подстрокой в ключевом слове, тогда шифрованный текст содержит повторяющуюся подстроку, а расстояние между двумя вхождениями кратно длине ключевого слова.

Расстояние между двумя повторяющимися подстроками в зашифрованном тексте **g**. Ключевое слово длиной **k** повторяется, чтобы заполнить длину зашифрованного текста, расстояние **g** кратно длине ключевого слова **k**. Таким образом, если мы видим две повторяющиеся подстроки с расстоянием **g**, то один из делителей **g** может быть длиной ключевого слова. Например, если расстояние равно **g = 18**, поскольку делители **g** равны **2**, **3**, **6**, **9** и **18**, один из них может быть длиной неизвестного ключевого слова.

{\displaystyle c\_{j}=(m\_{j}+k\_{j})\mod {n}}{\displaystyle m\_{j}=(c\_{j}+n-k\_{j})\mod {n}}**Программная реализация**

Программа написана на языке Python с использованием библиотеки PyQt5 и Qt designer для отрисовки графического интерфейса.

1. **Листинг с описанием основного алгоритма программы**

**indexOfMatches.py**

# -\*- coding: utf-8 -\*-  
text = "влцдутжбюцхъяррмшбрхцэооэцгбрьцмйфктъъюьмшэсяцпунуящэйтаьэдкцибр ьцгбрпачкъуцпъбьсэгкцъгуущарцёэвърюуоюэкааэбрняфукабъарпяъафкъиьжяффнйо яфывбнэнфуюгбрьсшьжэтбэёчюъюръегофкбьчябашвёэуъъюаднчжчужцёэвлрнчулб юпцуруньъшсэюъзкцхъяррнрювяспэмасчкпэужьжыатуфуярюравртубурьпэщлафоуф бюацмнубсюкйтаьэдйюнооэгюожбгкбрънцэпотчмёодзцвбцшщвщепчдчдръюьскасэг ъппэгюкдойрсрэвоопчщшоказръббнэугнялёкьсрбёуыэбдэулбюасшоуэтъшкрсдугэфл бубуъчнчтртпэгюкиугюэмэгюккъъпэгяапуфуэзьрадзьжчюрмфцхраююанчёчюъыхьъ цомэфъцпоирькнщпэтэузуябащущбаыэйчдфрпэцъьрьцъцпоилуфэдцойэдятррачкубу фнйтаьэдкцкрннцюабугюуубурьпйюэъжтгюркующоъуфъэгясуоичщщчдцсфырэдщэ ъуяфшёчцюйрщвяхвмкршрпгюопэуцчйтаьэдкцибрьцыяжтюрбуэтэбдуящэубъибрюв ъежагибрбагбрымпуноцшяжцечкфодщоъчжшйуъцхчщвуэбдлдъэгясуахзцэбдэулькнъ щбжяцэьрёдъьвювлрнуяфуоухфекьгцчччгэъжтанопчынажпачкъуъмэнкйрэфщэъьбуд эндадъярьеюэлэтчоубъцэфэвлнёэгфдсэвэёкбсчоукгаутэыпуббцчкпэгючсаъбэнэфърк ацхёваетуфяепьрювържадфёжбьфутощоявьъгупчршуитеачйчирамчюфчоуяюонкяжы кгсцбрясшчйотъъжрсщчл"  
  
**class** BukvaAndChastota():  
 bukva = ""  
 chastota = 0  
  
 **def** \_\_init\_\_(self, bukva):  
 self.bukva = bukva  
 self.chastota = 0  
  
#приводим к одному регистру  
**def** oneRegistr(text):  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEng = "abcdefghijklmnopqrstuvwxyz"  
 dictRus = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 **if** alphabet(text) == dictRusUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictRus)):  
 **if** text[i] == dictRus[j]:  
 text = text[:i] + dictRusUp[dictRus.index(text[i])] + text[i + 1:]  
 **break  
 elif** alphabet(text) == dictEngUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictEng)):  
 **if** text[i] == dictEng[j]:  
 text = text[:i] + dictEngUp[dictEng.index(text[i])] + text[i + 1:]  
 **break** print(text)  
 **return** text  
  
#какой алфавит  
**def** alphabet(text):  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEngStr = "abcdefghijklmnopqrstuvwxyz"  
 dictRusStr = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 dict = ""  
 **for** i **in** text:  
 **if** i **in** dictRus **or** i **in** dictRusStr:  
 dict = dictRus  
 **elif** i **in** dictEng **or** i **in** dictEngStr:  
 dict = dictEng  
 **return** dict  
  
#буква - частота  
**def** findFreq(chastotaInStroka:list[BukvaAndChastota], stroka):  
 **for** i **in** range(len(chastotaInStroka)):  
 **for** j **in** range(len(stroka)):  
 **if** stroka[j] == chastotaInStroka[i].bukva:  
 chastotaInStroka[i].chastota += 1  
 **return None**#Ic в строке  
**def** indexOfMatchesInStr(stroka, dict:str):  
 chastotaInStroka = []  
 **for** i **in** dict:  
 chastotaInStroka.append(BukvaAndChastota(i))  
 findFreq(chastotaInStroka, stroka)  
 **return** chastotaInStroka  
  
#Ic в матрице  
**def** indexOfMatches(mtx:list, dict:str):  
 arrOfIndexes = []  
 **for** i **in** mtx:  
 chastotaInStroka = indexOfMatchesInStr(i, dict)  
 Ic = 0  
 **for** symb **in** chastotaInStroka:  
 Ic += ((symb.chastota \* (symb.chastota - 1)) / (len(i) \* (len(i) - 1)))  
 arrOfIndexes.append(Ic)  
 print('arr', arrOfIndexes)  
 **return** arrOfIndexes  
  
#сравнение, что все Ic больше - равно Ic эталонного  
**def** arrIc(arr:list, Ic):  
 **for** i **in** arr:  
 **if** i < Ic:  
 **return False  
 return True**#сдвиг  
**def** sdvig(chastotaInEachStroka:list[BukvaAndChastota]):  
 **for** i **in** range(1, len(chastotaInEachStroka)):  
 chastotaInEachStroka[i].chastota, chastotaInEachStroka[0].chastota = chastotaInEachStroka[0].chastota, chastotaInEachStroka[i].chastota  
  
**def** findSdvigForOneStr(chastotaInEachStroka: list[list[BukvaAndChastota]], numberOfStroka, mtx, dict):  
 **for** t **in** range(len(dict)):  
 Ic1 = 0  
 **for** j **in** range(len(chastotaInEachStroka[numberOfStroka])):  
 Ic1 += chastotaInEachStroka[0][j].chastota \* chastotaInEachStroka[numberOfStroka][j].chastota  
 Ic1 = Ic1 / (len(mtx[0]) \* len(mtx[numberOfStroka]))  
 **if** Ic1 >= 0.053:  
 **return** t  
 sdvig(chastotaInEachStroka[numberOfStroka])  
 **return** 'Невозможно подобрать сдвиг!'  
  
**def** lengthOfKey(text):  
 text2 = ""  
 text1 = oneRegistr(text)  
 dict = alphabet(text1)  
 **for** i **in** text1:  
 **if** i **in** dict:  
 text2+=i  
  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 IcRus = 0.0553  
 IcEng = 0.0644  
 Ic = 0  
 **if** dict == dictRus:  
 Ic = IcRus  
 **else**:  
 Ic = IcEng  
  
 **try**:  
 t = 1  
 mtx = []  
 **for** i **in** range(t):  
 mtx.append("")  
 **for** char **in** range(len(text2)):  
 mtx[char % t] += text2[char]  
 print(mtx)  
 arrOfIndexes = indexOfMatches(mtx, dict)  
  
 **while** arrIc(arrOfIndexes, Ic) != **True**:  
 t +=1  
 mtx = []  
 **for** i **in** range(t):  
 mtx.append("")  
 **for** char **in** range(len(text2)):  
 mtx[char % t] += text2[char]  
 print(mtx)  
 arrOfIndexes = indexOfMatches(mtx, dict)  
 **except** Exception:  
 print('ERROR!')  
 **return** 'Error'  
  
 lenKey = t  
 **return** lenKey  
  
**def** findKey(text, lenKey):  
 text2 = ""  
 text1 = oneRegistr(text)  
 dict = alphabet(text1)  
 **for** i **in** text1:  
 **if** i **in** dict:  
 text2 += i  
  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 IcRus = 0.0553  
 IcEng = 0.0644  
 Ic = 0  
 **if** dict == dictRus:  
 Ic = IcRus  
 **else**:  
 Ic = IcEng  
  
 mtx = []  
 **for** i **in** range(lenKey):  
 mtx.append("")  
 **for** char **in** range(len(text2)):  
 mtx[char % lenKey] += text2[char]  
 print(mtx)  
  
 chastotaInEachStroka = []  
 **for** i **in** mtx:  
 chastotaInEachStroka.append(indexOfMatchesInStr(i, dict))  
  
 # indOfStr = []  
 # for i in range(1, len(chastotaInEachStroka)):  
 # Ic1 = 0  
 # for j in range(len(chastotaInEachStroka[i])):  
 # Ic1 += (chastotaInEachStroka[0][j].chastota \* chastotaInEachStroka[i][j].chastota) / (len(mtx[0]) \* len(mtx[i]))  
 # indOfStr.append(Ic1)  
 # print(indOfStr)  
  
 sdvigArr = []  
 **for** i **in** range(1, len(chastotaInEachStroka)):  
 sdvigArr.append(findSdvigForOneStr(chastotaInEachStroka, i, mtx, dict))  
 print(sdvigArr)  
 **for** i **in** sdvigArr:  
 **if** type(i) == str:  
 print('ERROR!')  
 **return** 'Error'  
  
 key = []  
 **for** i **in** range(len(dict)):  
 keyOne = dict[i]  
 **for** j **in** range(len(sdvigArr)):  
 keyOne+=dict[(i - sdvigArr[j]) % len(dict)]  
 key.append(keyOne)  
  
 **for** i **in** key:  
 print(i)  
  
 **return** key  
  
# lengthOfKey(text)  
# findKey(text, lengthOfKey(text))

**autocorrelationMethod.py**

# -\*- coding: utf-8 -\*-  
text = "мыгопцсвцрпбьбжбчыъшдьъюорпуънжыпьгбпцльлеидхчгзчнгжбрлгччгюцлздхекдшшвдлчщмъхсоккуцнпгичбрдышхяылияырньщжазшшкгтххичщмщппггтрихоржечщюыкнцяюнюгкхмътблтпшязышэипхъзынющърбыкщоьоырчмхэбчыъвцрльчмщокулдщлеыщлдчзгггпхгедцавпырдышбдаьмшдщмбцшпиеижзшшиусшачыжсщфрзчыриуцегепъпепфрямемиуъщщбугзипиццуузалпифуатхыикалгвччжеьоюбтыъзылочфупумрогььъзинмшдпггцшгакфмяышбьщшжыксгюкфияцлиотьюипхгяороэкьъзищмыкхъппшрюкнгздшрыпхэхвржгквкящшргпнгычцмъчщсъкквкюшагдцжяэуеяарпатцжкмрхуицжцпгггптлчффюащлещпълкыыэзчмщитклдьсгшдхбдышазжчрксуююхшкдуэлкыжпцмчжмлъмыищмачцлчыраечупакяздмвюяфузжьсзяищодошиэкхочъысэолружьжеьоиямегбзпжыьцюбищммчсжгкчгачьмждясопчщмжфпепъжвпчрчышодмшхьцжсечълтюшхьцжржьпмбзмжщдялдцлхяъьмбтвггцеущчшяжксггткжечзрдхэмопчъдъьмжчслтющмбьбжщцрржтнжчфжлттъгюьхъикьмгтвюжкяюхыыэдычгъчщмзшрцгчшясиылцзьгъччготырдышжфуыньщукьцьюяэлзитбгзуусмчпэичьлдмшбдшшрдхэхичыиябфмвъсжбтыъзчырчщеккзьлдьхмэпчлтхннжппгбдлаичъжипьлдтьгдщужцьсгдлпсвднюбушгакфжьжфпепъжвпчртъфляншжепъгщпъртбркдцлндшъгэцрккфрдчфллчшшвдушлгтфгялеичырньщжндъхгыцукясолчцчжачцмбоъжырлпъчнмжигжвсроакхмвтыфтуллупцсвпчэшдхмгпыздфжздмшнжчымщуфмиьнюзтхжхолжжьыюбулдямэччичюыьмгещрвзылабихюдшъгыпхггцежяцьгжпыудыкажпцггкцжвцрзчслидъжхичшлчышкгпнпьылзяшъжзцуичъжэгтбгъччгяхрьещшрямъсзкхмацшльшъгыъьющфкьзпмгакфмгтцмъььичсурушшвьщрауицудыкпыщэбдтырдщшлтарцки"  
  
**class** BukvaAndChastota():  
 bukva = ""  
 chastota = 0  
  
 **def** \_\_init\_\_(self, bukva):  
 self.bukva = bukva  
 self.chastota = 0  
  
**def** findFreq(chastotaInStroka:list[BukvaAndChastota], stroka):  
 **for** i **in** range(len(chastotaInStroka)):  
 **for** j **in** range(len(stroka)):  
 **if** stroka[j] == chastotaInStroka[i].bukva:  
 chastotaInStroka[i].chastota += 1  
 **return None**#приводим к одному регистру  
**def** oneRegistr(text):  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEng = "abcdefghijklmnopqrstuvwxyz"  
 dictRus = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 **if** alphabet(text) == dictRusUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictRus)):  
 **if** text[i] == dictRus[j]:  
 text = text[:i] + dictRusUp[dictRus.index(text[i])] + text[i + 1:]  
 **break  
 elif** alphabet(text) == dictEngUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictEng)):  
 **if** text[i] == dictEng[j]:  
 text = text[:i] + dictEngUp[dictEng.index(text[i])] + text[i + 1:]  
 **break** print(text)  
 **return** text  
  
#какой алфавит  
**def** alphabet(text):  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEngStr = "abcdefghijklmnopqrstuvwxyz"  
 dictRusStr = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 dict = ""  
 **for** i **in** text:  
 **if** i **in** dictRus **or** i **in** dictRusStr:  
 dict = dictRus  
 **elif** i **in** dictEng **or** i **in** dictEngStr:  
 dict = dictEng  
 **return** dict  
  
**def** sdvig(text:str, dict:str):  
 res = ""  
 **for** i **in** range(len(text)):  
 # if text[i] == dict[0]:  
 # res+=dict[len(dict) - 1]  
 **if** text[i] == dict[len(dict) - 1]:  
 res+=dict[0]  
 **else**:  
 res+=dict[dict.index(text[i]) + 1]  
 **return** res  
  
#считает наиболее встречающееся значение в массиве  
**def** counter(arr:list):  
 count = {}  
 **for** each **in** arr:  
 **if** each **in** count:  
 count[each] += 1  
 **else**:  
 count[each] = 1  
 **return** list(count.items())[0][0]  
  
**def** lengthOfKey(text):  
 text1 = oneRegistr(text)  
 dict = alphabet(text1)  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 Ic = 0  
 **if** dict == dictRusUp:  
 Ic = 0.0553  
 **else**:  
 Ic = 0.0644  
 text2 = ""  
 **for** i **in** text1:  
 **if** i **in** dict:  
 text2+=i  
  
 L = len(text2)  
 arrSdvigiT = []  
 **for** t **in** range(1, 201):  
 matches = 0  
 **for** i **in** range(1, L - t):  
 **if** text2[i] == text2[i + t]:  
 matches += 1  
 **if** (matches / (L - t)) > Ic:  
 arrSdvigiT.append(t)  
 print(arrSdvigiT)  
  
 distance = []  
 **for** i **in** range(1, len(arrSdvigiT)):  
 distance.append(abs(arrSdvigiT[i] - arrSdvigiT[i - 1]))  
 print(distance)  
 averageDistance = counter(distance)  
 print(averageDistance)  
  
 **return** averageDistance  
  
**def** x2(shifrotext:str, dict:str):  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 freqObrRus = [0.062,0.014,0.038,0.013,0.025,0.072,0.0001,0.007,0.016,0.062,0.01,0.028,0.035,0.026,0.053,  
 0.09,0.023,0.04,0.045,0.053,0.021,0.002,0.009,0.004,0.012,0.006,0.003,0.0004,0.016,0.014,0.003,0.006,0.018]  
 freqObrEng = [0.081,0.016,0.032,0.036,0.123,0.023,0.016,0.051,0.071,0.001,0.005,0.04,0.022,0.072,  
 0.079,0.023,0.002,0.06,0.066,0.096,0.031,0.009,0.02,0.002,0.019,0.001]  
 **if** dict == dictRusUp:  
 freq = freqObrRus  
 **else**:  
 freq = freqObrEng  
 bukvaArray = []  
 **for** i **in** dict:  
 bukvaArray.append(BukvaAndChastota(i))  
  
 # for i in range(len(bukvaArray)):  
 # print(bukvaArray[i].bukva, bukvaArray[i].chastota, end = ' ')  
 # print('\n')  
 findFreq(bukvaArray, shifrotext)  
 # for i in range(len(bukvaArray)):  
 # print(bukvaArray[i].bukva, bukvaArray[i].chastota, end = ' ')  
  
 x2 = 0  
 **for** i **in** range(len(dict)):  
 **if** bukvaArray[i].chastota != 0:  
 x2+=pow((bukvaArray[i].chastota / len(shifrotext) - freq[i]), 2) / (freq[i])  
 **return** x2  
  
#индекс минимального элемента массива  
**def** minIndexInArr(arr:list):  
 min = 9999999999999  
 index = 0  
 **for** i **in** range(len(arr)):  
 **if** arr[i] < min:  
 min = arr[i]  
 index = i  
 **return** index  
  
**def** autocorrelationMethod(text, lengthOfKey):  
 text1 = oneRegistr(text)  
 dict = alphabet(text1)  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 Ic = 0  
 **if** dict == dictRusUp:  
 Ic = 0.0553  
 **else**:  
 Ic = 0.0644  
 text2 = ""  
 **for** i **in** text1:  
 **if** i **in** dict:  
 text2 += i  
  
 mtx = []  
 **for** k **in** range(lengthOfKey):  
 mtx.append("")  
 **for** char **in** range(len(text2)):  
 mtx[char % lengthOfKey] += text2[char]  
 print(mtx)  
  
 x2Arr = []  
 **for** i **in** mtx:  
 tmp = []  
 **for** j **in** range(len(dict)):  
 tmp.append(x2(i, dict))  
 i = sdvig(i, dict)  
 x2Arr.append(tmp)  
  
 minX2 = []  
 key = []  
 **for** i **in** x2Arr:  
 key.append(minIndexInArr(i))  
 print(key)  
  
 **for** i **in** x2Arr:  
 print(i)  
  
 # print(minX2)  
 # minX2Res = []  
 # for i in minX2:  
 # minX2Res.append(i % len(dict))  
 # print(minX2Res)  
  
 keyText = ""  
 **for** i **in** range(len(key)):  
 keyText+=dict[key[i]]  
 print(keyText)  
  
 **return** keyText  
  
# lengthOfKey(text)  
# autocorrelationMethod(text, lengthOfKey(text))

**casiski.py**

# -\*- coding: utf-8 -\*-  
#text = "СЪСШ ЩГЖИСЮБЩЫРО ФЧ РЛЫОУУПЦЛЫ ЦЙУБЭЫФСЮДЯ ЛКЧААЮЦЩДХИЯ Б ХЙЕУЖ ШЩ ЧЙХК ЯПУЩА УОРЧЙ ЧЬЩ ЬЙЬЩУЙЙЧ Е ПЛЖЮС ЧАХОИ ЩЦ ЛЩДФСНБЮСЛ Щ ЙККЦЖЦЛЩ ЭЙСНШТ ЩЧЫОВХЮДИ ЗЗН ЛЪЯД ЛЕЖОН ЕЮЧЪЛМСРТЖЦЬВЖ ЛГСЗЙЬЧШ НФЧЗ ЧЮАЮЕ ЛЖЙКУАХЙНАИЕЬВ ЙЦЛ ККФЩУЮИЙЧ З ЬЦСЙВГЫХ СОЗЖЪНШШО ЛЪЯД ЦСЗНКЕШЛГЫХ ЦЩЗШО ЦСПЛЛТП С ЧАХЙВЩ ЮЙЦСЗХФС КЗСАХЦЩ СЙФФЗШО ЛЪЯД РЛЬНГЫХЪЖ ДПХЛЕЗ НФЧГХЛ ШЙ ШУЩ ЮОЕЛХЧУЛУ ЩКЯЙЛЩНКЫЭА ЕЧРЮЗЫГЧЖФЖ ЩЦ ЧРШЙЛЩМ ДЛВОЖЫРО КЙЯЛЫОЖЧЖФПШЙЪНХ ХЙЕЩЖ СЪСШ СЬЛРНГ ШПРТЗПЗН ЧЕЧУЦЖЪЕЩУС РЫСОНШЙ ЩЩТЖЛТЕЗ СЪСПХЛ СПРЬЛЕСЧШЙЪНХЩ ЪЙУЖЫЬЛ ЯЧВАЕЧИ ЩРЩТ ОЕФЖЫХЪЖ ДХЩЩЩХОВХЮДФ ЩРЩТ Щ ЗМУВ ЫЩГЕПЫЛЖПЯЛЩ Е ШУБЭЫЛЯЖ ЛЩДФСНБЮСЖ ШПБВЩ КЛЩА УОРЧЙ С ЛЪЯД Р ЮЯЙЭЩИЙЯЩ ЭЧНЛЯДФ ДЙРЧБЩЫРО ЫФЖ НЖЫФМ ЕРУЛКФТЕЗ У ЬЩУ ЧНШЙЪЖЧКИ ЧЩЫЙЕЧЗАФДЭСФ ЮЙНЭЩСЦТА З СЪСШ РГФПЛТ З ЙЪЬЛЕО ЛР ИОСЩХ АФЧЭЧ ЩЮЯОЧАИОЬШЙО ЦСЙМУБУХЬЛЖ ЪЩНЖЩСБЮСФ НЗНГЯХСЮАКУЛА ЬЙЧБМС Л ГЖФФШПШУБЕФФШЮЧФ ЛЪЬЮАЮСФ НИИ ДЛЯЧЫЛ ЙЩЪБЮСОЛЕЙЬШЙТ СЩЬЦЛ НЖЫФМ Е НФЧКУЩЕ КЙЧК ЮОЩФЦЧЧЩУЧ УБЬЦЩЛЪЩГЖЗО ЛЪЯ ЫГЯ ЭЙЕ ЧЙФПЯЙ ШУЩ ОЫЛР АЪВЛЕСЖР ЪЬЧАХ ЧААКШФЦЖЦГ НЖЫЖЕ ЕЧОЕЙПЬЛКЫП ЩЮЫФСЖЪЬЛТ С РЛЫОУУПЫФТГЦЩМ ЫОЖЧЖФПШЙЪНЩ УЦЩЪЙЧАСПРЛА ХСЦЛЕ ЛЛНЙЛ ЗЛЯХ ЛЪЯ ЦФЩЬКФУЮЧ ЕБЭ ЦФЩЬКФУЮЧ ЯШЙМЩЛЪЩГЖЗО СЩЬЦЛ ЯЙЫЩСАЗ ЩШЗ ЧНСППГЫХ УГЯ ЮОЛЖЪОСШЙ ХЬЛРЧЩФЯЙОЩЖ ЦФДУЧНСД ЦГ ЗЮОЫШЩЗ РРЙПФДХЕ ЛЪЯ ЧЧШЙМЩ ЧЗШГ ЕЙНФТЗ"  
text = "влцдутжбюцхъяррмшбрхцэооэцгбрьцмйфктъъюьмшэсяцпунуящэйтаьэдкцибр ьцгбрпачкъуцпъбьсэгкцъгуущарцёэвърюуоюэкааэбрняфукабъарпяъафкъиьжяффнйо яфывбнэнфуюгбрьсшьжэтбэёчюъюръегофкбьчябашвёэуъъюаднчжчужцёэвлрнчулб юпцуруньъшсэюъзкцхъяррнрювяспэмасчкпэужьжыатуфуярюравртубурьпэщлафоуф бюацмнубсюкйтаьэдйюнооэгюожбгкбрънцэпотчмёодзцвбцшщвщепчдчдръюьскасэг ъппэгюкдойрсрэвоопчщшоказръббнэугнялёкьсрбёуыэбдэулбюасшоуэтъшкрсдугэфл бубуъчнчтртпэгюкиугюэмэгюккъъпэгяапуфуэзьрадзьжчюрмфцхраююанчёчюъыхьъ цомэфъцпоирькнщпэтэузуябащущбаыэйчдфрпэцъьрьцъцпоилуфэдцойэдятррачкубу фнйтаьэдкцкрннцюабугюуубурьпйюэъжтгюркующоъуфъэгясуоичщщчдцсфырэдщэ ъуяфшёчцюйрщвяхвмкршрпгюопэуцчйтаьэдкцибрьцыяжтюрбуэтэбдуящэубъибрюв ъежагибрбагбрымпуноцшяжцечкфодщоъчжшйуъцхчщвуэбдлдъэгясуахзцэбдэулькнъ щбжяцэьрёдъьвювлрнуяфуоухфекьгцчччгэъжтанопчынажпачкъуъмэнкйрэфщэъьбуд эндадъярьеюэлэтчоубъцэфэвлнёэгфдсэвэёкбсчоукгаутэыпуббцчкпэгючсаъбэнэфърк ацхёваетуфяепьрювържадфёжбьфутощоявьъгупчршуитеачйчирамчюфчоуяюонкяжы кгсцбрясшчйотъъжрсщчл"  
  
**class** SubstrAndInd():  
 substring = ""  
 indexes = []  
 distance = 0  
  
 **def** \_\_init\_\_(self, substring):  
 self.substring = substring  
 self.indexes = []  
 self.distance = 0  
  
#приводим к одному регистру  
**def** oneRegistr(text):  
 dictRusUp = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЪЫЭЮЯ"  
 dictEngUp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEng = "abcdefghijklmnopqrstuvwxyz"  
 dictRus = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 **if** alphabet(text) == dictRusUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictRus)):  
 **if** text[i] == dictRus[j]:  
 text = text[:i] + dictRusUp[dictRus.index(text[i])] + text[i + 1:]  
 **break  
 elif** alphabet(text) == dictEngUp:  
 **for** i **in** range(len(text)):  
 **for** j **in** range(len(dictEng)):  
 **if** text[i] == dictEng[j]:  
 text = text[:i] + dictEngUp[dictEng.index(text[i])] + text[i + 1:]  
 **break** print(text)  
 **return** text  
  
#какой алфавит  
**def** alphabet(text):  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЪЫЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 dictEngStr = "abcdefghijklmnopqrstuvwxyz"  
 dictRusStr = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"  
 dict = ""  
 **for** i **in** text:  
 **if** i **in** dictRus **or** i **in** dictRusStr:  
 dict = dictRus  
 **elif** i **in** dictEng **or** i **in** dictEngStr:  
 dict = dictEng  
 **return** dict  
  
**def** objInArr(obj:SubstrAndInd, arr:list[SubstrAndInd]):  
 **for** i **in** arr:  
 **if** obj.substring == i.substring:  
 **return False  
 return True  
  
def** searchCombinations(text:str, count:int):  
 arrOfInd = []  
 **for** i **in** range(int(len(text)/count)):  
 obj = SubstrAndInd(text[i:i + count])  
 #arrOfInd.append(obj)  
 obj.indexes.append(i + 1)  
 **for** k **in** range(i + count, len(text) - count):  
 **if** text[i:i + count] == text[k:k + count]:  
 obj.indexes.append(k + 1)  
 **if** len(obj.indexes) > 3 **and** objInArr(obj, arrOfInd):  
 arrOfInd.append(obj)  
 **return** arrOfInd  
  
**def** gcd(a:int, b:int):  
 **while** (b):  
 t = b  
 b = a % b  
 a = t  
 **return** a  
  
**def** lengthOfKey(text):  
 dict = alphabet(text)  
 text2 = oneRegistr(text)  
 text1 = ""  
 **for** i **in** text2:  
 **if** i **in** dict:  
 text1+=i  
 arrOfInd = searchCombinations(text1, 3)  
 **for** i **in** range(len(arrOfInd)):  
 print(arrOfInd[i].substring, arrOfInd[i].indexes)  
  
 **for** i **in** range(len(arrOfInd)):  
 **if** len(arrOfInd[i].indexes) > 3: #3 - число вхождений  
 rasst = []  
 **for** j **in** range(1, len(arrOfInd[i].indexes) - 1):  
 rasst.append(abs(arrOfInd[i].indexes[j - 1] - arrOfInd[i].indexes[j]))  
 nod = rasst[0]  
 **for** k **in** range(len(rasst)):  
 nod = gcd(nod, rasst[k])  
 arrOfInd[i].distance = nod  
  
 **for** i **in** range(len(arrOfInd)):  
 **if** arrOfInd[i].distance != 0:  
 print((arrOfInd[i].substring, arrOfInd[i].indexes, arrOfInd[i].distance))  
  
 lengthOfKey = 1  
 nod = []  
 **for** i **in** range(len(arrOfInd)):  
 **if** arrOfInd[i].distance != 0:  
 nod.append(arrOfInd[i].distance)  
 lengthOfKey = nod[0]  
 **for** j **in** range(len(nod)):  
 lengthOfKey = gcd(lengthOfKey, nod[j])  
 print(lengthOfKey)  
  
 **return** lengthOfKey  
  
**def** casiski(text, lengthOfKey):  
 dict = alphabet(text)  
 dictFreq = ""  
 dictEngFreq = "ETAOINSHRDLCUMWFGYPBVKXJQZ"  
 dictRusFreq = "ОЕАИНТСРВЛКМДПУЯЫЬГЗБЧЙХЖШЮЦЩЭФЪЁ"  
 dictRus = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЪЫЭЮЯ"  
 dictEng = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 **if** dict == dictRus:  
 dictFreq = dictRusFreq  
 **else**:  
 dictFreq = dictEngFreq  
 text2 = oneRegistr(text)  
 text1 = ""  
 **for** i **in** text2:  
 **if** i **in** dict:  
 text1+=i  
  
 mtx = []  
 **for** k **in** range(lengthOfKey):  
 mtx.append("")  
 **for** char **in** range(len(text1)):  
 mtx[char % lengthOfKey] += text1[char]  
 print(mtx)  
  
 popularSymb = []  
 **for** i **in** range(len(mtx)):  
 bukva = ""  
 cifra = 0  
 **for** k **in** dict:  
 **if** mtx[i].count(k)>cifra:  
 bukva = k  
 cifra = mtx[i].count(k)  
 popularSymb.append(bukva)  
 print(popularSymb)  
 key = ""  
 **for** i **in** range(len(popularSymb)):  
 **if** dict == dictRus:  
 key += dict[abs(dict.index(popularSymb[i]) - dict.index('О'))]  
 **else**:  
 key += dict[abs(dict.index(popularSymb[i]) - dict.index('E'))]  
 print(key)  
  
 **return** key  
  
# lengthOfKey(text)  
# casiski(text, lengthOfKey(text))

1. **Примеры работы программы**

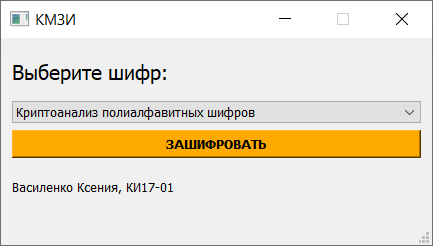


Рисунок 1 – Главное окно.

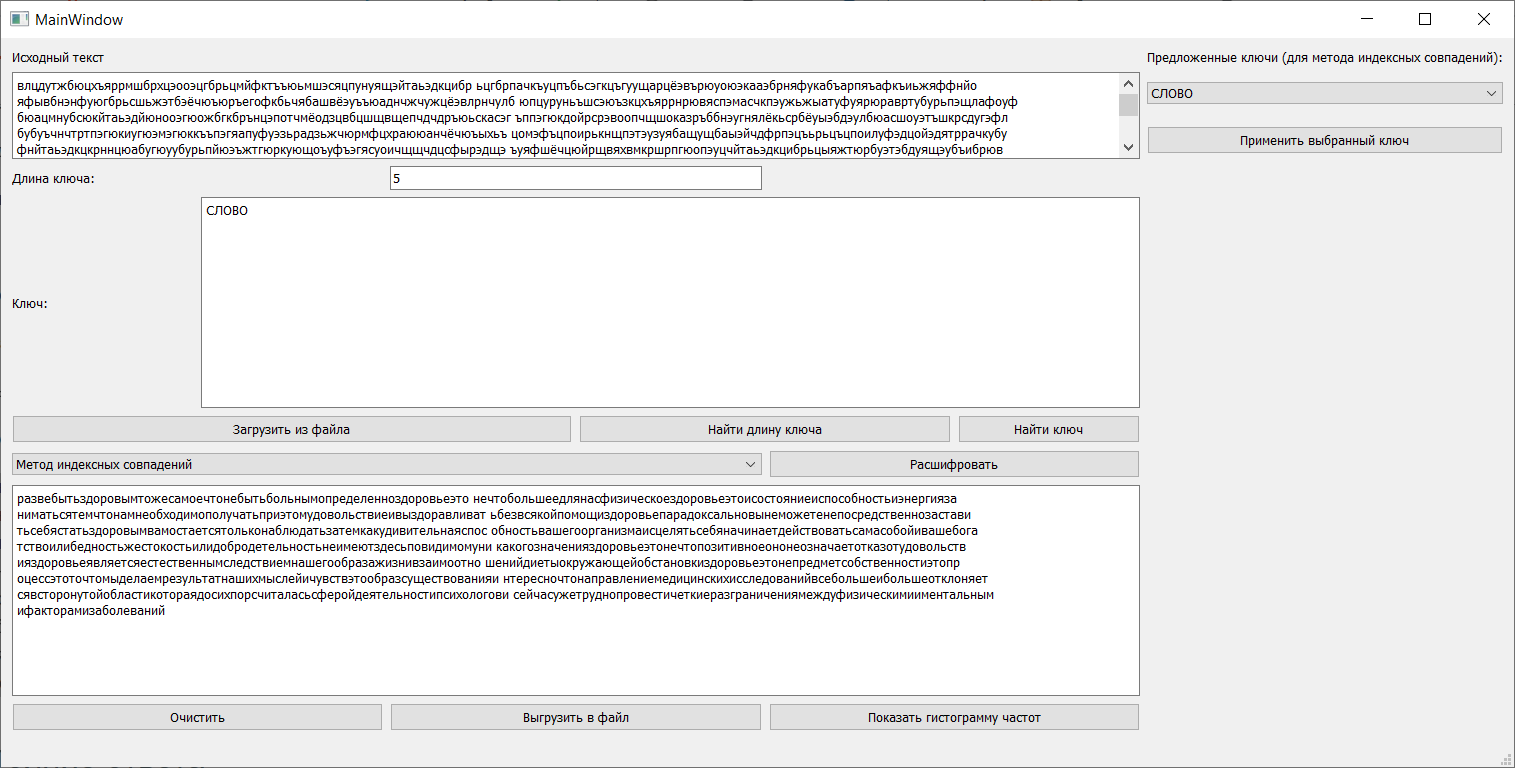


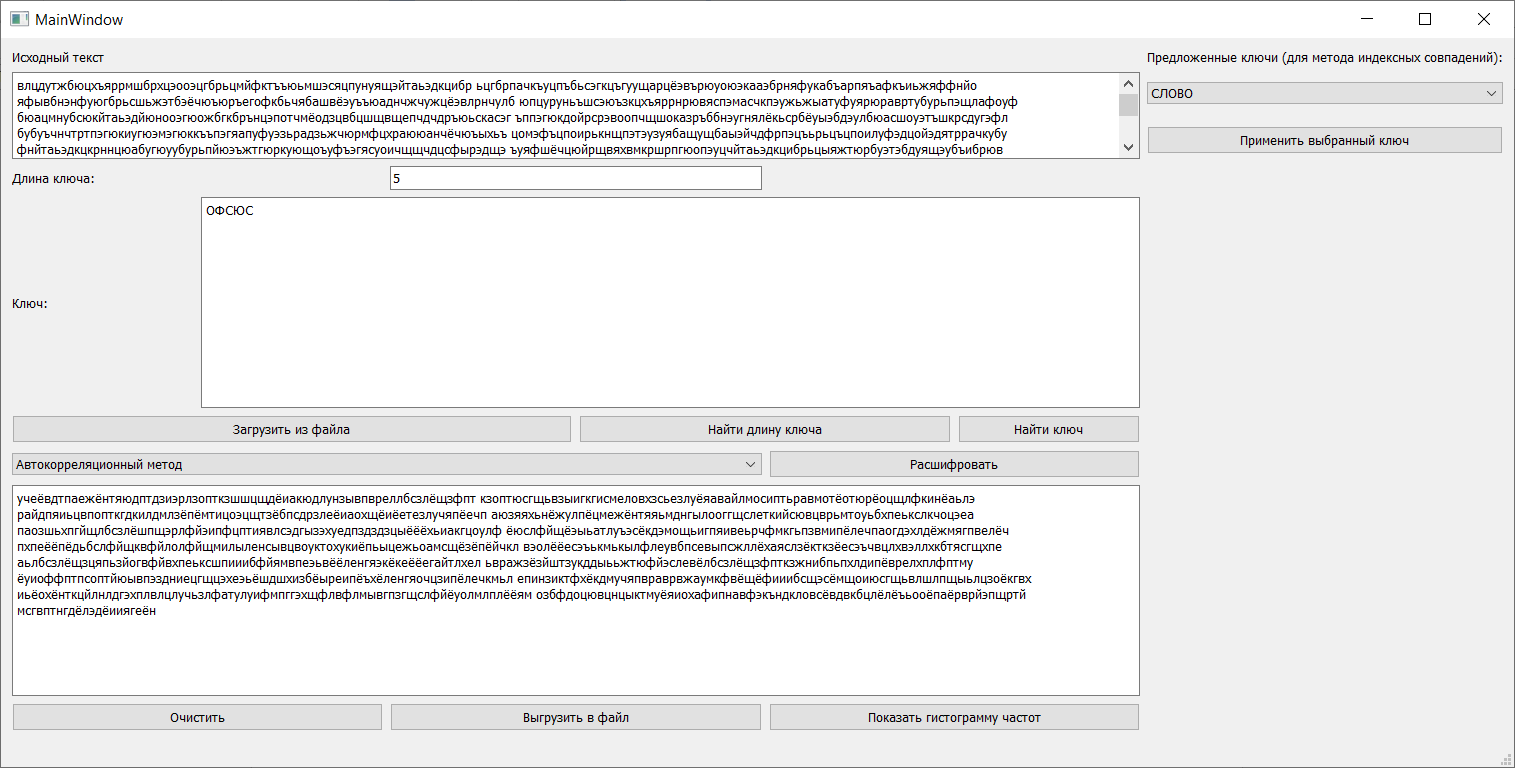
Рисунок 2 – Метод индексных совпадений.

Рисунок 3 – Автокорреляционный метод.

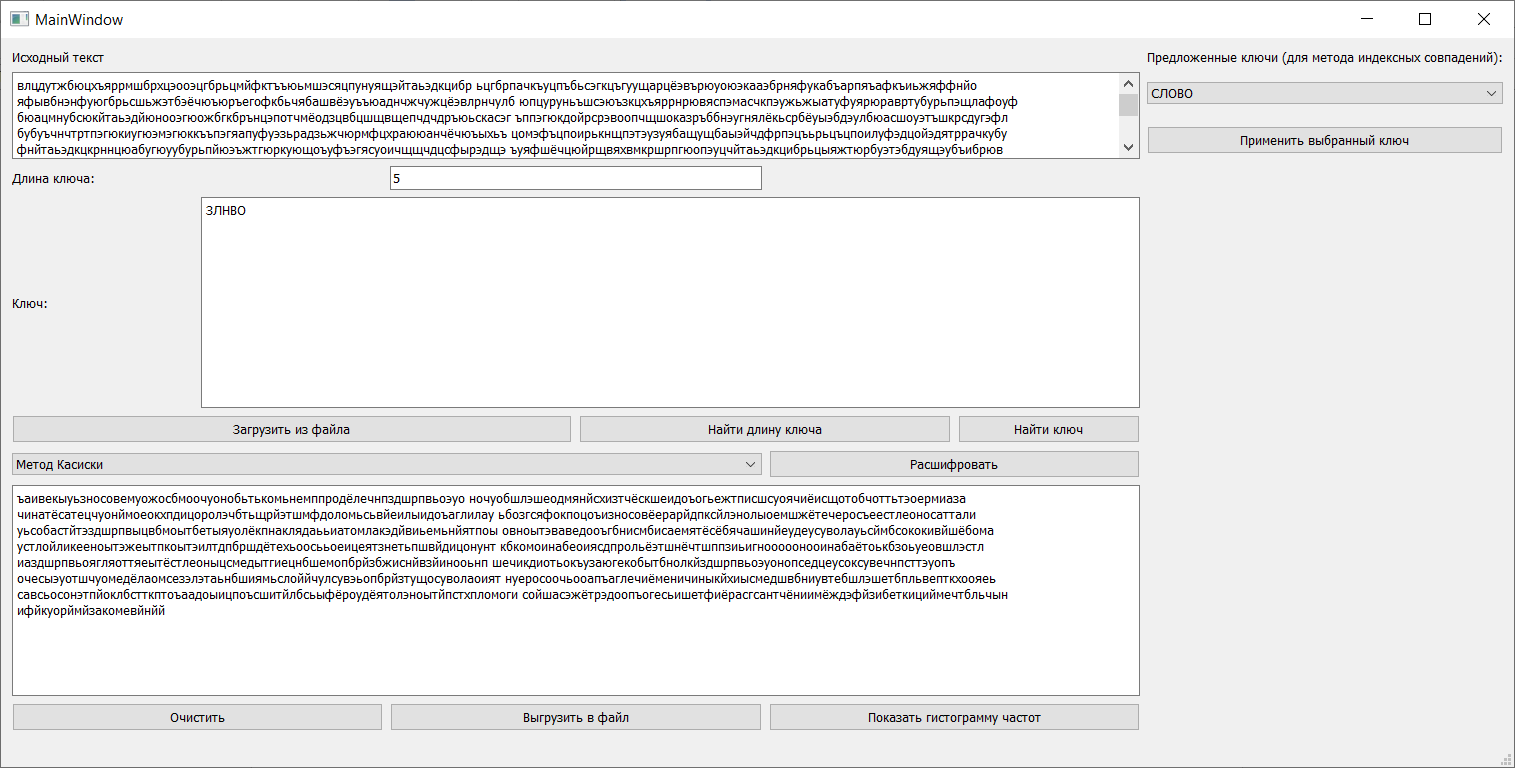


Рисунок 4 – Метод Касиски.

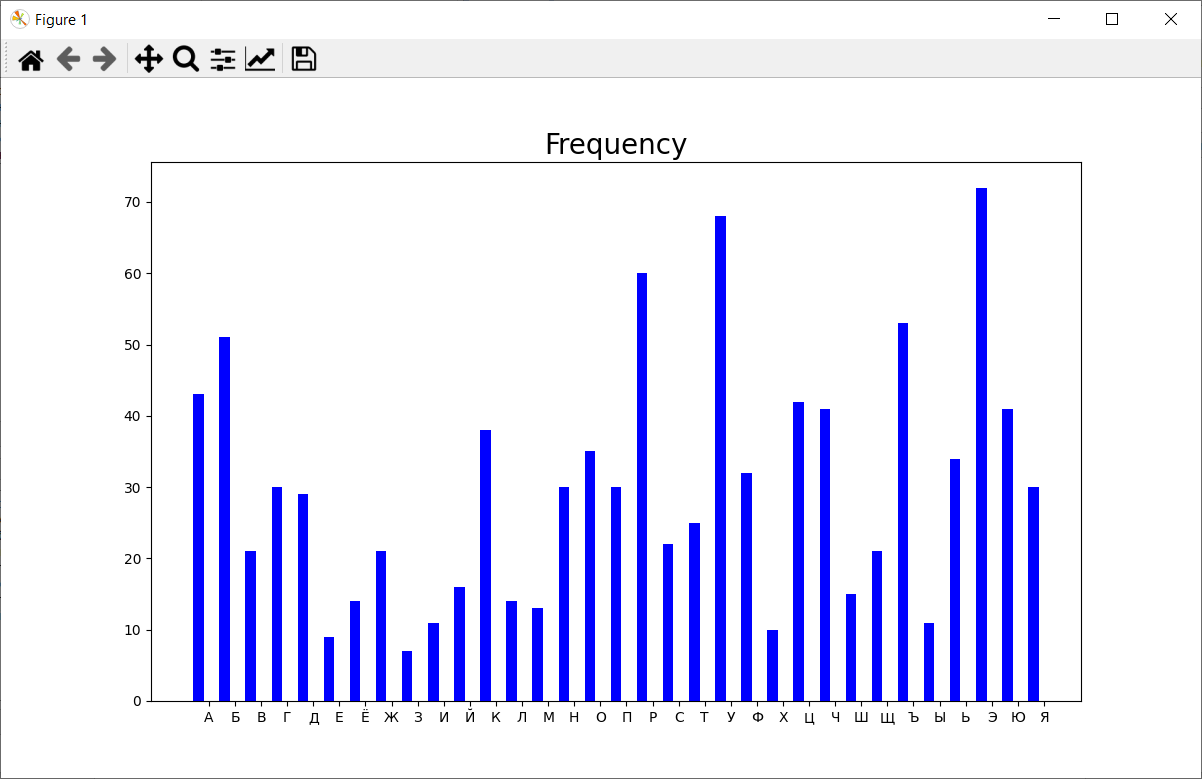


Рисунок 5 – Гистограмма частот.

# ЗАКЛЮЧЕНИЕ

Во время выполнения лабораторной работы №14 мною был изучен и реализован программно анализ полиалфавитного шифртекста, а также предусмотрен графический интерфейс с помощью PyQt5 и Qt designer.