Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 3
по дисциплине «Криптографические методы защиты информации»
ТЕМА РАБОТЫ

Подстановочные шифры

Студент гр. МКБ231
Е.В. Шараев
«27» мая 2024 г.

Руководитель
Заведующий кафедрой информационной безопасности киберфизических систем
канд. техн. наук, доцент
_____О.О. Евсютин
«___» _____ 2024 г.

Москва 2024

СОДЕРЖАНИЕ

1 Задание на практическую работу.....3

2 Краткая теоретическая часть.....4

3 Описание программной реализации.....4

3.1 Шифр простой замены ……...5

3.2 Аффинный шифр ...…….7

3.3 Аффинный рекуррентный шифр ……....……..9

4 Примеры зашифрования / расшифрования ……..…..11

5 Криптоанализ подстановочных шифров …….…….11

6 Выводы о проделанной работе.....11

7 Приложение 1 11

8 Приложение 2 12

9 Приложение 3 19

1 Задание на практическую работу

Целью данной работы является приобретение навыков программной реализации и ручного шифрования подстановочных шифров.

В рамках практической работы необходимо выполнить следующее:

1 написать программную реализацию следующих шифров:

- шифр простой замены
- аффинный шифр
- аффинный рекуррентный шифр

2 изучить методы криптоанализа моноалфавитных подстановочных шифров с использованием дополнительных источников

3 провести криптоанализ данных шифров;

4 подготовить отчет о выполненной работе

Аффинный шифр и аффинный рекуррентный шифр должны быть реализованы одним из двух рассмотренных выше способов:

1. в кольце классов вычетов Z_m;
2. в поле Галуа F_{p^n}.

Студент самостоятельно выбирает способ реализации.

Программа должна обладать следующей функциональностью для каждого из реализованных в ней шифров:

- принимать на вход произвольную последовательность символов, вводимую пользователем в качестве открытого текста или шифртекста;
- принимать на вход секретный ключ вида, соответствующего конкретному шифру;
- осуществлять зашифрование или расшифрование введенного текста по выбору пользователя.

Отчет должен содержать следующие составные части:

- ✓ раздел с заданием;
- ✓ раздел с краткой теоретической частью;
- ✓ раздел с двумя-тремя примерами «ручного» шифрования для произвольных последовательностей символов;
- ✓ раздел с результатами работы программы для тех же последовательностей символов, что и в предыдущем разделе;

- ✓ раздел с примерами криптоанализа реализованных шифров;
- ✓ раздел с выводами о проделанной работе.

2 Краткая теоретическая часть

2.1 Шифр простой замены

Шифр простой замены, простой подстановочный шифр, моноалфавитный шифр — класс методов шифрования, которые сводятся к созданию по определённому алгоритму таблицы шифрования, в которой для каждой буквы открытого текста существует единственная сопоставленная ей буква шифр-текста. Само шифрование заключается в замене букв согласно таблице. Для расшифровки достаточно иметь ту же таблицу, либо знать алгоритм, по которому она генерируется.

К шифрам простой замены относятся многие способы шифрования, возникшие в древности или средневековье, как, например, Атбаш (также читается как этбаш) или шифр Цезаря. Для вскрытия подобных шифров используется частотный криптоанализ.

2.2 Аффинный шифр

В аффинном шифре каждой букве алфавита размера m ставится в соответствие число из диапазона 0… m-1. Затем при помощи специальной формулы, вычисляется новое число, которое заменит старое в шифртексте. где x — номер шифруемой буквы в алфавите; m — размер алфавита; a, b — ключ шифрования. Этот шифр, также как и шифр простой замены поддается простому частотному криптоанализу.

2.3 Аффинный рекуррентный шифр

Аффинный рекуррентный шифр похож на простой Аффинный, но в рекуррентном шифре для каждой буквы, начиная с третьей, ключи составляются новые. Новые ключи рассчитываются по формуле:

$$\alpha_i = \alpha_{i-1} * \alpha_{i-2}$$

$$\beta_i = \beta_{i-1} + \beta_{i-2}$$

Таким образом для зашифрования каждого символа в аффинном рекуррентном шифре используется свой уникальный ключ. По этой причине данный шифр взлому частотным анализом не поддается так как один и тот же символ может быть зашифрован разными символами шифртекста и наоборот, два разных символа могут быть зашифрованы одним и тем же.

3. Описание программной реализации

Все файлы данной практической работы я опубликовал в своем гитхаб репозитории https://github.com/Djoongaar/simple_cipher

Программная реализация целиком находится в файле https://github.com/Djoongaar/simple_cipher/blob/master/simple_cipher.py

Максимально подробные комментарии к коду я постарался оставить в самих блоках кода, а здесь лишь описание основных методов класса.

Демонстрационный юпитер ноутбук (пдф файл этого ноутбука приложен к данному отчету) https://github.com/Djoongaar/simple_cipher/blob/master/demo.ipynb

Юпитер ноутбук с примерами криптоанализа (пдф файл этого ноутбука приложен к данному отчету) https://github.com/Djoongaar/simple_cipher/blob/master/decrypt.ipynb

3.1 Программная реализации шифра простой замены

Шифр простой замены реализован в классе SimpleCipher в файле simple_cipher.py

```
class SimpleCipher:
    """ Шифр простой замены """

    def __init__(self, mapping=None):
        """
        self.alphabet - Используемый алфавит (ascii + пробел)
        self.alphabet_map - Таблица соответствия идекса и букв алфавита
        self.secret_key - Ключ шифрования - таблица замены. Если Таблица
        замены не задана то используется таблица замены по-умолчанию.
        """
        self.alphabet = string.ascii_lowercase + string.ascii_uppercase + string.digits + string.punctuation + " "
        self.module = len(self.alphabet)
        self.alphabet_map = self.__get_alphabet_map()
        self.secret_key = self.__get_mapping(mapping)
```

В качестве алфавита шифрования (self.alphabet) был установлен алфавит ascii в нижнем регистре, верхнем регистре, цифры, знаки препинания и пробел.

Модулем алгоритма (self.module) взят размер алфавита.

В качестве ключа шфирования принята таблица замены (self.alphabet_map) - два массива (прямой и инверсионный), которые зашифровывают порядковый номер символа открытого текста в шифртекст. Класс подразумевает также возможность передачи ему на вход своей таблицы замен. Размерность такой таблицы должна при этом соответствовать модулю алгоритма (self.module).

```
@staticmethod
def __get_mapping(mapping):
    if mapping is None:
        mapping = [
            [29, 44, 65, 31, 48, 4, 92, 84, 10, 37, 90, 9, 66, 93, 25, 17, 47, 35, 33, 20, 23, 19, 63, 30, 24, 75,
             68, 27, 1, 62, 59, 76, 7, 50, 16, 0, 5, 26, 22, 8, 85, 80, 54, 74, 89, 39, 36, 57, 55, 18, 40, 69, 43,
             28, 86, 71, 91, 6, 78, 87, 12, 41, 15, 34, 56, 70, 58, 52, 21, 2, 60, 82, 67, 45, 49, 53, 11, 88, 73,
             13, 79, 42, 46, 64, 61, 38, 94, 77, 83, 72, 14, 81, 51, 32, 3],
            [35, 28, 69, 94, 5, 36, 57, 32, 39, 11, 8, 76, 60, 79, 90, 62, 34, 15, 49, 21, 19, 68, 38, 20, 24, 14,
             37, 27, 53, 0, 23, 3, 93, 18, 63, 17, 46, 9, 85, 45, 50, 61, 81, 52, 1, 73, 82, 16, 4, 74, 33, 92, 67,
             75, 42, 48, 64, 47, 66, 30, 70, 84, 29, 22, 83, 2, 12, 72, 26, 51, 65, 55, 89, 78, 43, 25, 31, 87, 58,
             80, 41, 91, 71, 88, 7, 40, 54, 59, 77, 44, 10, 56, 6, 13, 86]
        ]
    return mapping
```

Зашифрования и расшифрование описано в методах encrypt и decrypt. По сути они просто проверяют порядковый номер элемента шифртекста или открытого текста и возвращают номер элемента в который он должен быть трансформирован. Затем склеивают все символы в одну строку и возвращают её.

```
    def encrypt(self, message: str):
        encrypted_message = []

        for i in message:
            index = self._get_index_by_letter(i)
            enc_index = self.secret_key[0][index]
            encrypted_message.append(self._get_letter_by_index(enc_index))

        return ''.join(encrypted_message)

    def decrypt(self, encrypted_message: str):
        decrypted_message = []

        for i in encrypted_message:
            enc_index = self._get_index_by_letter(i)
            decr_index = self.secret_key[1][enc_index]
            decrypted_message.append(self._get_letter_by_index(decr_index))

        return ''.join(decrypted_message)
```

*

3.2 Программная реализации аффинного шифра

Аффинный шифр реализован в классе AffineCipher, который наследуется от класса SimpleCipher, однако при инициации переопределяет параметр ключа шифрования self.secret_key, так как он у Аффинного шифра откличается.

```
class AffineCipher(SimpleCipher):
    """ Аффинный шифр """

    def __init__(self, secret_key=None):
        super().__init__()
        self.secret_key = self._get_secret_key(secret_key)
```

Ключ шифрования в Аффинном шифре должен состоять из пары целочисленных значений (a, b), причем `a` должно быть взаимнопростым с модулем алгоритма , `b` - производльное число. Оба значения должны быть меньше модуля алгоритма.

```
    def _get_secret_key(self, secret_key):
        """
        Генератор ключа афинного шифра.
        m - модуль алгоритма (мощность алфавита шифрования)
        a - должен быть взаимнопростым с модулем шифрования
        b - произвольное число (в пределах модуля алгоритма)
        :return: (a, b)
        """
        if secret_key is None:
            a = random.randrange(1, 100) % self.module

            if a < 3 or math.gcd(a, self.module) != 1:
                a = random.randrange(1, 100) % self.module

            b = random.randrange(1, 100) % self.module
        else:
            a, b = secret_key

        return a, b
```

Зашифрование по сути является вычислением порядкового номера элемента шифртекста в алфавите. Порядковый номер вычисляется по формуле: $y_i = a * x_i + b$
Затем все элементы шифртекста конкатенируются в одну строку и возвращаеются в виде результата.

```
def encrypt(self, message: str):
    """
    Производит преобразования аффинного шифра:
    y = (ax + b) % m
    :param message: str - Открытый текст для зашифрования
    :return: encrypted_message: str - Зашифрованный текст
    """
    encrypted_message = []

    for i in message:
        index = self._get_index_by_letter(i)
        enc_index = (self.secret_key[0] * index + self.secret_key[1]) % self.module
        encrypted_message.append(self._get_letter_by_index(enc_index))

    return ''.join(encrypted_message)
```

Расшифрование представляет собой процедуру обратную зашифрованию. Из порядкового номера элемента шифртекста вычитается `b`, затем результат делится на `a`. Поскольку в полях классов вычетов операция деления не предусмотрена, она заменяется на умножение на обратный элемента `a_inv`

```
def decrypt(self, encrypted_message: str):
    """
    Производит преобразования расшифрования аффинного шифра
    x = ((y - b) * a_inv) % m
    :param encrypted_message: str - Строка с зашифрованным текстом
    :return: decrypted_message: str - Строка с расшифрованным текстом
    """
    decrypted_message = []

    for i in encrypted_message:
        index = self._get_index_by_letter(i)
        a_inv = pow(self.secret_key[0], -1, self.module)
        dec_index = ((index - self.secret_key[1]) * a_inv) % self.module
        decrypted_message.append(self._get_letter_by_index(dec_index))

    return ''.join(decrypted_message)
```

Полученное число — это номер элемента в алфавите. Затем все элементы склеиваются в строку и возвращается расшифрованный текст.

3.3 Программная реализации аффинного рекуррентного шифра

Аффинный рекуррентный шфир релизован в классе AffineRecurrentCipher, унаследованного от класса AffineCipher (аффинный шифр). В рекуррентном аффинном шифре ключь шифрования также представлен парой цифр (a,b), однако ключей шифрования два. Они используются для шифрования первого и второго символов открытогт текста.

```
class AffineRecurrentCipher(AffineCipher):
    """ Аффинный рекуррентный шифр """

    def __init__(self, secret_key=None, secret_key_2=None):
        super().__init__()
        self.secret_keys = [
            self._get_secret_key(secret_key),
            self._get_secret_key(secret_key_2)
        ]
```

Далее, начиная с третьего элемента, `a` и `b` пересчитываются, и получается для для шифрования каждого элемента открытогт текста расчитываются своя пара ключей. Логика пересчета ключей реализована в методе _override_key и она выполняется на каждом шаге шифрования начиная с третьего элемента

```
def _override_key(self, secret_keys):
    """ Пересчитываем ключ шифрования """
    a = (secret_keys[-1][0] * secret_keys[-2][0]) % self.module
    b = (secret_keys[-1][1] + secret_keys[-2][1]) % self.module
    return a, b
```

Зашифрование реализовано также как и в Аффинном шифре, за исключением генерации нового ключа на каждом шаге начиная с третьего.

```
def encrypt(self, message: str):
    secret_keys = [self.secret_keys[0], self.secret_keys[1]]
    encrypted_message = []

    for num, i in enumerate(message):
        if num > 1:
            # Пересчитываем ключ шифрования и добавляем в массив
            a, b = self._override_key(secret_keys)
            secret_keys.append((a, b))
        else:
            a, b = secret_keys[num]

        # Шифруем очередной символ открытого текста
        index = self._get_index_by_letter(i)
        enc_index = (a * index + b) % self.module
        encrypted_message.append(self._get_letter_by_index(enc_index))

    return ''.join(encrypted_message)
```

Расшифрование в рекуррентном аффинном шифре выполняется так же как и в аффинном шифре, за исключением генерации нового ключа на каждом шаге расшифрования начиная с третьего элемента.

```
def decrypt(self, encrypted_message: str):
    secret_keys = [self.secret_keys[0], self.secret_keys[1]]
    decrypted_message = []

    for num, i in enumerate(encrypted_message):
        if num > 1:
            # Пересчитываем ключ шифрования и добавляем в массив
            a, b = self._override_key(secret_keys)
            secret_keys.append((a, b))
        else:
            a, b = secret_keys[num]

        # Расшифровываем очередной символ
        index = self._get_index_by_letter(i)
        a_inv = pow(a, -1, self.module)
        decr_index = (index - b) * a_inv % self.module
        decrypted_message.append(self._get_letter_by_index(decr_index))

    return ''.join(decrypted_message)
```

4 Примеры зашифрования / расшифрования

Примеры зашифрования и расшифрования программным способом и вручную приведены в Приложении 2

5 Криптоанализ подстановочных шифров

Примеры дешифрации (криптоанализа) рассматриваемых подстановочных шифров приведены в Приложении 3

6. Выводы

Глубоко изучил алгоритмы подстановочных шифров, методы их криптоанализа, а также получил практические навыки по программной реализации таких методов шифрования.

ПРИЛОЖЕНИЕ 1.

Список использованных источников

1. Исходный код использованный программы — URL: https://github.com/Djoongaar/simple_cipher/blob/master/simple_cipher.py

2. Исходный код демонстрации работы программы и ручного примера зашифрования / расшифрования — URL: https://github.com/Djoongaar/simple_cipher/blob/master/demo.ipynb

3. Исходный код примеров криптоанализа шифртекстов шифров замены — URL: https://github.com/Djoongaar/simple_cipher/blob/master/decrypt.ipynb

4. Статья Википедии о Аффинном шифре — URL: https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%84%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80

5. Статья Википедии о Аффинном шифре — URL: https://studbooks.net/1786378/informatika/affinnyy_rekurrentnyy_shifr

Приложение 2. Демонстрация работы программы

Часть 1. Шифр простой замены

1.1 Программное зашифрование / расшифрование

In [1]:
```
# Програма шифра простой замены реализована
# в модуле simple_cipher классе SimpleCipher

!ls -l
```

```
total 680
-rw-r--r-- 1 evgeny evgeny 140344 May 19 19:02 decrypt.ipynb
-rw-r--r-- 1 evgeny evgeny  30772 May 27 18:47 demo.ipynb
-rw-r--r-- 1 evgeny evgeny  26671 May 19 16:44 english.jpg
drwxr-xr-x 2 evgeny evgeny   4096 May 19 17:53 __pycache__
-rw-r--r-- 1 evgeny evgeny   7998 May 19 17:53 simple_cipher.py
drwxr-xr-x 4 evgeny evgeny   4096 May 17 14:52 venv
-rw-r--r-- 1 evgeny evgeny 471411 May 28 17:53 отчет.docx
```

In [2]:
```
# Импортирую класс для демонстрации его работы

from simple_cipher import SimpleCipher
```

In [3]:
```
# Объявлю объект класса и выведу на экран его атрибуты

encryptor = SimpleCipher()

# Алфавит шифрования
encryptor.alphabet
```

Out[3]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [4]:
```
# Модуль алгоритма шифрования
encryptor.module
```

Out[4]:
```
95
```

In [5]:
```
# Так как классу не был передан ключ шифрования (Таблица замены)
# то объект создался с ключом по-умолчанию
print(encryptor.secret_key)
```

```
[[29, 44, 65, 31, 48, 4, 92, 84, 10, 37, 90, 9, 66, 93, 25, 17, 47, 35, 33, 20, 23, 19, 63, 30, 24, 75, 6
8, 27, 1, 62, 59, 76, 7, 50, 16, 0, 5, 26, 22, 8, 85, 80, 54, 74, 89, 39, 36, 57, 55, 18, 40, 69, 43, 28,
86, 71, 91, 6, 78, 87, 12, 41, 15, 34, 56, 70, 58, 52, 21, 2, 60, 82, 67, 45, 49, 53, 11, 88, 73, 13, 79,
42, 46, 64, 61, 38, 94, 77, 83, 72, 14, 81, 51, 32, 3], [35, 28, 69, 94, 5, 36, 57, 32, 39, 11, 8, 76, 60,
79, 90, 62, 34, 15, 49, 21, 19, 68, 38, 20, 24, 14, 37, 27, 53, 0, 23, 3, 93, 18, 63, 17, 46, 9, 85, 45, 5
0, 61, 81, 52, 1, 73, 82, 16, 4, 74, 33, 92, 67, 75, 42, 48, 64, 47, 66, 30, 70, 84, 29, 22, 83, 2, 12, 7
2, 26, 51, 65, 55, 89, 78, 43, 25, 31, 87, 58, 80, 41, 91, 71, 88, 7, 40, 54, 59, 77, 44, 10, 56, 6, 13, 8
6]]
```

In [6]:
```
# Создам новый ключ (таблицу замены) и передам
# его объекту класса для шифрования

# import random

# Вот например таблица случайных размещений чисел от 0 до модуля
# random_nums = random.sample(range(encryptor.module), encryptor.module)
# print(random_nums)

random_nums = [
    58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6,
    87, 36, 39, 55, 81, 84, 88, 56, 80, 7, 23, 52, 83, 21, 73, 76, 82, 77,
    78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 62,
    46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74,
    44, 24, 72, 35, 1, 11, 64, 0, 38, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57,
    10, 70, 12, 4]

print(random_nums)
```

```
[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88, 56, 80,
7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 62,
46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 35, 1, 11, 64, 0, 38, 4
2, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4]
```

In [7]:
```
# Теперь создан обратный массив random_nums_inv для
# более удобного обратного преобразования

# random_nums_inv = [0 for _ in range(encryptor.module)]

# for num, i in enumerate(random_nums):
#     random_nums_inv[i] = num

random_nums_inv = [
    79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52,
    89, 70, 45, 31, 51, 28, 73, 6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75,
    19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11,
    49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88,
    74, 32, 71, 55, 33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63,
    69, 15, 4, 40, 1]

print(random_nums_inv)
```

```
[79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73, 6, 16,
65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11,
49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88, 74, 32, 71, 55, 33, 35, 36, 84, 26, 2
2, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40, 1]
```

In [8]:
```
# Теперь оба массива можно передать классу SimpleCipher

encryptor = SimpleCipher([random_nums, random_nums_inv])
```

In [9]:
```
# Проверим что класс принял новый ключ для шифрования данных

print(encryptor.secret_key)
```

```
[[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88, 56, 8
0, 7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 69, 22, 17, 6
2, 46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 35, 1, 11, 64, 0, 3
8, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4], [79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 9
3, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73, 6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 2
0, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 64, 5, 29, 11, 49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7,
38, 68, 50, 92, 88, 74, 32, 71, 55, 33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40,
1]]
```

In [10]:
```
# Кажется что всё правильно!
# Приступим к зашифрованию

# Определяем текст для шифрования

open_text = "I have a Dream!"

# Шифруем
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[10]:
```
'?e%63}e6e0g}69I'
```

In [11]:
```
# Расшифруем
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
I have a Dream!
```

Out[11]:
```
True
```

1.2 Зашифрование / расшифрование вручную

Итак, имеем:

- Открытый текст ***"I have a Dream!"*** который требуется зашифровать.
- Алфиавит

  ```
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
  ```

- Ключ шифрования в виде матрицы замены символов

  ```
  [[58, 94, 86, 65, 92, 51, 25, 66, 33, 2, 49, 53, 61, 47, 59, 91, 26, 6, 87, 36, 39, 55, 81, 84, 88,
  56, 80, 7, 23, 52, 83, 21, 73, 76, 82, 77, 78, 41, 67, 37, 93, 29, 13, 30, 85, 20, 3, 48, 15, 54, 6
  9, 22, 17, 62, 46, 75, 45, 28, 16, 9, 14, 32, 34, 89, 50, 27, 40, 63, 68, 90, 19, 74, 44, 24, 72, 3
  5, 1, 11, 64, 0, 38, 42, 5, 43, 79, 31, 8, 60, 71, 18, 57, 10, 70, 12, 4],
  [79, 76, 9, 46, 94, 82, 17, 27, 86, 59, 91, 77, 93, 42, 60, 48, 58, 52, 89, 70, 45, 31, 51, 28, 73,
  6, 16, 65, 57, 41, 43, 85, 61, 8, 62, 75, 19, 39, 80, 20, 66, 37, 81, 83, 72, 56, 54, 13, 47, 10, 6
  4, 5, 29, 11, 49, 21, 25, 90, 0, 14, 87, 12, 53, 67, 78, 3, 7, 38, 68, 50, 92, 88, 74, 32, 71, 55,
  33, 35, 36, 84, 26, 22, 34, 30, 23, 44, 2, 18, 24, 63, 69, 15, 4, 40, 1]]
  ```

- Для удобства также буду пользоваться таблицей индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Для зашифрования требуется заменить каждый исмвол из открытого текста символом из того же алфавита в соответствии с нулевым вектором матрицы замены символов. Например, если первый символ открытого текста I имеет индекс: 34, тогда в соответствии с матрицей замен символов я шифрую его символом с индексом 82, то есть символом ? Следующие символы открытого текста меняю по тому же алгоритму:

- пробел имеет индекс 94 => 4, то есть => e
- h имеет индекс 7 => 66, то есть => %
- a имеет индекс 0 => 58, то есть => 6
- v имеет индекс 21 => 55, то есть => 3
- e имеет индекс 4 => 92, то есть => }
- пробел имеет индекс 94 => 4, то есть => e
- a имеет индекс 0 => 58, то есть => 6
- пробел имеет индекс 94 => 4, то есть => e
- D имеет индекс 29 => 52, то есть => 0
- r имеет индекс 17 => 6, то есть => g
- e имеет индекс 4 => 92, то есть => }
- a имеет индекс 0 => 58, то есть => 6
- m имеет индекс 12 => 61, то есть => 9
- ! имеет индекс 62 => 34, то есть => I

Сложив все символы в строку получаем **'?e%63}e6e0g}69I'**, что соответствует программно зашифрованной строке.

Расшифрование производится в соответствии с вторым вектором матрицы замены символов. Символ ? в алфавите имеет идекс 82. В первом векторе матрицы по индексу 82 размещена цифра 34, то есть смвол I. Оставшие символы расшифровываются по тому же алгоритму:

- e имеет индекс 4 => 94, то есть => пробел
- % имеет индекс 66 => 7, то есть => h
- 6 имеет индекс 58 => 0, то есть => a
- 3 имеет индекс 55 => 21, то есть => v
- } имеет индекс 92 => 4, то есть => e
- e имеет индекс 4 => 94, то есть => пробел
- 6 имеет индекс 58 => 0, то есть => a
- e имеет индекс 4 => 94, то есть => пробел
- 0 имеет индекс 52 => 29, то есть => D
- g имеет индекс 6 => 17, то есть => r
- } имеет индекс 92 => 4, то есть => e
- 6 имеет индекс 58 => 0, то есть => a
- 9 имеет индекс 61 => 12, то есть => m
- I имеет индекс 34 => 62, то есть => !

Сложив все символы в строку получить **I have a Dream!**

Часть 2. Аффинный шифр

2.1 Программное зашифрование / расшифрование

In [12]:
```
# Программа аффинного шифра реализована
# в модуле simple_cipher классе AffineCipher

from simple_cipher import AffineCipher

# Объявлю объект класса и выведу на экран его атрибуты

encryptor = AffineCipher()

# Алфавит шифрования класс AffineCipher унаследовал от SimpleCipher
# как впрочем и все остальные его атрибуты

encryptor.alphabet
```

Out[12]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [13]:
```
# В том числе и модуль
encryptor.module
```

Out[13]: 95

In [14]:
```
# А вот ключ шифрования тут другой
# В аффинном шифровании ключ - это коэффициенты a, b
# необходимые для аффинного преобразрования ax + b,
# где x - это закодированный символ, например его
# порядковый номер в принятом алфавите

# Так как ключ не был передан классу при его объявлении
# то он был автоматически сгененерирован.

encryptor.secret_key
```

Out[14]: (56, 74)

In [15]:
```
# Также ключ шифрования можно передать снаружи

encryptor = AffineCipher((62, 74))
encryptor.secret_key
```

Out[15]: (62, 74)

In [16]:
```
# Попробуем зашифровать какой-то текст
# Определяем еще один текст для шифрования

open_text = "Dum spiro spero!"

# Зашифрую на тех же ключах
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[16]: '&<6mY2a@^mY2L@^x'

In [17]:
```
# Расшифруем
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
Dum spiro spero!
```

Out[17]: True

2.2 Зашифрование / расшифрование вручную

Дано:

- Открытый текст ***"Dum spiro spero!"***
- Алфиавит

  `abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~`

- Ключ шифрования в виде пары чисел

```
(62, 74)
```

- Модуль алгоритма 95
- Таблица индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Аффинный шифр с помощью коэффициентов a и b меняет выбирает новый символ из того же словаря для зашифрования:

i - индекс символа из открытого текста

$\hat{i} = (ai + b)$ % m - индекс символа шифртекста

Тогда, для зашифрования требуется выполнить следующие операции над каждым сиволом:

- D - индекс символа 29 => $(62 * 29 + 74) mod 95$ => 67 => &
- u - индекс символа 20 => $(62 * 20 + 74) mod 95$ => 79 => <
- m - индекс символа 12 => $(62 * 12 + 74) mod 95$ => 58 => 6

- пробел - индекс символа 94 => $(62 * 94 + 74) mod 95$ => 12 => m
- s - индекс символа 18 => $(62 * 18 + 74) mod 95$ => 50 => Y
- p - индекс символа 15 => $(62 * 15 + 74) mod 95$ => 54 => 2
- i - индекс символа 8 => $(62 * 8 + 74) mod 95$ => 0 => a
- r - индекс символа 17 => $(62 * 17 + 74) mod 95$ => 83 => @
- o - индекс символа 14 => $(62 * 14 + 74) mod 95$ => 87 => ^
- пробел - индекс символа 94 => $(62 * 94 + 74) mod 95$ => 12 => m
- s - индекс символа 18 => $(62 * 18 + 74) mod 95$ => 50 => Y
- p - индекс символа 15 => $(62 * 15 + 74) mod 95$ => 54 => 2
- e - индекс символа 4 => $(62 * 4 + 74) mod 95$ => 37 => L
- r - индекс символа 17 => $(62 * 17 + 74) mod 95$ => 83 => @
- o - индекс символа 14 => $(62 * 14 + 74) mod 95$ => 87 => ^
- ! - индекс символа 62 => $(62 * 62 + 74) mod 95$ => 23 => x

Собрав все символы в строку получим шифртекст **'&<6mY2a@^mY2L@^x'**

Для расшифрования шифртекста требуется найти элемент обратный a (a - должно быть обратимым по модулю алгоритма)

$a^{-1} = 23$, а затем вычислить индекс исходного элемента открытого текста выполнив операции обратные зашифрованию

$i = (\hat{i} - b) * a^{-1}$ % m - индекс символа шифртекста

Тогда, для расшифрования требуется выполнить следующие операции над каждым сиволом:

- & => имеет индекс 67 => (67 - 74) * 23 mod 95 = 29 => D
- < => имеет индекс 79 => (79 - 74) * 23 mod 95 = 20 => u
- 6 => имеет индекс 58 => (58 - 74) * 23 mod 95 = 12 => m
- m => имеет индекс 12 => (12 - 74) * 23 mod 95 = 94 => пробел
- Y => имеет индекс 50 => (50 - 74) * 23 mod 95 = 18 => s
- 2 => имеет индекс 54 => (54 - 74) * 23 mod 95 = 15 => p
- a => имеет индекс 0 => (0 - 74) * 23 mod 95 = 8 => i
- @ => имеет индекс 83 => (83 - 74) * 23 mod 95 = 17 => r
- ^ => имеет индекс 87 => (87 - 74) * 23 mod 95 = 14 => o
- m => имеет индекс 12 => (12 - 74) * 23 mod 95 = 94 => пробел
- Y => имеет индекс 50 => (50 - 74) * 23 mod 95 = 18 => s
- 2 => имеет индекс 54 => (54 - 74) * 23 mod 95 = 15 => p
- L => имеет индекс 37 => (37 - 74) * 23 mod 95 = 4 => e
- @ => имеет индекс 83 => (83 - 74) * 23 mod 95 = 17 => r
- ^ => имеет индекс 87 => (87 - 74) * 23 mod 95 = 14 => o
- x => имеет индекс 23 => (23 - 74) * 23 mod 95 = 62 => !

Сложив все символы в строку получим **Dum spiro spero!**

Часть 3. Аффинный рекуррентный шифр

3.1 Программное зашифрование / расшифрование

In [18]:
```
# Программа аффинного рекуррентного шифра реализована
# в модуле simple_cipher классе AffineRecurrentCipher

from simple_cipher import AffineRecurrentCipher

# Объявлю объект класса и выведу на экран его атрибуты

encryptor = AffineRecurrentCipher()

# Алфавит шифрования класс AffineRecurrentCipher унаследовал от AffineCipher
# как впрочем и все остальные его атрибуты

encryptor.alphabet
```

Out[18]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

In [19]:
```
# В том числе и модуль
encryptor.module
```

Out[19]:
```
95
```

In [20]:
```
# Но для зашифрования и расшифрования аффинный рекуррентный шифр
# использует другие ключи. У него их изначально два, но на
# каждом шаге шифрования (начиная со второго блока) генерируется
# новый на основе двух предыдущих

encryptor.secret_keys
```

Out[20]:
```
[(61, 14), (67, 93)]
```

In [21]:
```
# Чтобы не пользоваться ключами по-умолчанию алгоритму можно
# передать свои ключи, например вот так:

encryptor = AffineRecurrentCipher((39, 54), (18, 46))
encryptor.secret_keys
```

Out[21]:
```
[(39, 54), (18, 46)]
```

In [22]:
```
# Попробуем зашифровать какой-то текст
# Определяем еще один текст для шифрования

open_text = "This is the way!"

# Зашифрую на тех же ключах
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[22]:
```
'e:q(txk!|TX}H#GE'
```

In [23]:
```
# Расшифрую и сравню с исходным текстом
decrypted_text = encryptor.decrypt(encrypted_text)
print(decrypted_text)
bool(decrypted_text == open_text)
```

```
This is the way!
```

Out[23]:
```
True
```

3.2 Зашифрование / расшифрование вручную

Дано:

- Открытый текст ***"This is the way!"***
- Алфиавит

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
```

- Ключи шифрования в виде двух пар чисел

```
[(39, 54), (18, 46)]
```

- Модуль алгоритма 95
- Таблица индексов символов алфавита

```
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9, 'k': 10, 'l': 11,
'm': 12, 'n': 13, 'o': 14, 'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19, 'u': 20, 'v': 21, 'w': 22,
'x': 23, 'y': 24, 'z': 25, 'A': 26, 'B': 27, 'C': 28, 'D': 29, 'E': 30, 'F': 31, 'G': 32, 'H': 33,
'I': 34, 'J': 35, 'K': 36, 'L': 37, 'M': 38, 'N': 39, 'O': 40, 'P': 41, 'Q': 42, 'R': 43, 'S': 44,
'T': 45, 'U': 46, 'V': 47, 'W': 48, 'X': 49, 'Y': 50, 'Z': 51, '0': 52, '1': 53, '2': 54, '3': 55,
'4': 56, '5': 57, '6': 58, '7': 59, '8': 60, '9': 61, '!': 62, '"': 63, '#': 64, '$': 65, '%': 66,
'&': 67, "'": 68, '(': 69, ')': 70, '*': 71, '+': 72, ',': 73, '-': 74, '.': 75, '/': 76, ':': 77,
';': 78, '<': 79, '=': 80, '>': 81, '?': 82, '@': 83, '[': 84, '\\': 85, ']': 86, '^': 87, '_': 88,
'`': 89, '{': 90, '|': 91, '}': 92, '~': 93, ' ': 94}
```

Также как и в обычном аффинном шифре в аффинном рекуррентном шифре для шифрования используется формула

$\hat{i} = (ai + b)$ % m, однако коэффициента a и b на каждом символе разные.

Пересчет коэффициентов выполняется по формуле:

$a_i = a_{i-1} * a_{i-2}$ % m

$b_i = b_{i-1} + b_{i-2}$ % m

Зашифруем строку "This is the way!"

- T => i = 45, a = 39, b = 54 => $\hat{i}$ = (39*45+54)%95 = 4 => e
- h => i = 7, a = 18, b = 46 => $\hat{i}$ = (18*7+46)%95 = 77 => :
- i => i = 8, a = 37, b = 5 => $\hat{i}$ = (37*8+5)%95 = 16 => q
- s => i = 18, a = 1, b = 51 => $\hat{i}$ = (18+51)%95 = 69 => (
- пробел => i = 94, a = 37, b = 56 => $\hat{i}$ = (94*37+56)%95 = 19 => t
- i => i = 8, a = 37, b = 12 => $\hat{i}$ = (8*37+42)%95 = 23 => x
- s => i = 18, a = 39, b = 68 => $\hat{i}$ = (18+23)%95 = 10 => k
- пробел => i = 94, a = 18, b = 80 => $\hat{i}$ = (94+65)%95 = 62 => !
- t => i = 19, a = 37, b = 53 => $\hat{i}$ = (19+88)%95 = 91 => |
- h => i = 7, a = 1, b = 38 => $\hat{i}$ = (7+38)%95 = 45 => T
- e => i = 4, a = 37, b = 91 => $\hat{i}$ = (4*37+91)%95 = 49 => X
- пробел => i = 94, a = 37, b = 34 => $\hat{i}$ = (94*37+34)%95 = 92 => }

- w => i = 22, a = 39, b = 30 => $\hat{i}$ = (22*39+30)%95 = 33 => H
- a => i = 0, a = 18, b = 64 => $\hat{i}$ = (0+64)%95 = 64 => #
- y => i = 24, a = 37, b = 94 => $\hat{i}$ = (24*37+94)%95 = 32 => G
- ! => i = 62, a = 1, b = 63 => $\hat{i}$ = (62+63)%95 = 30 => E

Сложим все символы в одну строку и получим 'e:q(txk!|TX}H#GE'

Для расшифровки, так же как и в аффинном шифре требуется вычислить a^{-1}, но поскольку a на каждом шаге меняется то и вычислять его нужно каждый раз заново. Кэффициент b такой же как при вычислениях выше. Расшифрование производится по формуле:

$$i = (\hat{i} - b) * a^{-1}$$

- e => $\hat{i}$ = 4 , a^{-1} = 39 => (4-54)*39 % 95 = 45 => T
- : => $\hat{i}$ = 77, a^{-1} = 37 => (77-46)*37 % 95 = 7 => h
- q => $\hat{i}$ = 16, a^{-1} = 18 => (16-5)*18 % 95 = 8 => i
- (=> $\hat{i}$ = 69, a^{-1} = 1 => (69 - 51)*1 % 95 = 18 => s
- t => $\hat{i}$ = 19, a^{-1} = 18 => (19 - 56)*18 % 95 = 94 => пробел
- x => $\hat{i}$ = 23, a^{-1} = 18 => (23 - 12)*18 % 95 = 8 => i
- k => $\hat{i}$ = 10, a^{-1} = 39 => (10 - 68)*39 % 95 = 18 => s
- ! => $\hat{i}$ = 62, a^{-1} = 37 => (62 - 80)*37 % 95 = 94 => пробел
- | => $\hat{i}$ = 91, a^{-1} = 18 => (91 - 53)*18 % 95 = 19 => t
- T => $\hat{i}$ = 45, a^{-1} = 1 => (45 - 38)*1 % 95 = 7 => h
- X => $\hat{i}$ = 49, a^{-1} = 18 => (49 - 91)*18 % 95 = 4 => e
- } => $\hat{i}$ = 92, a^{-1} = 18 => (92 - 34)*18 % 95 = 94 => пробел
- H => $\hat{i}$ = 33, a^{-1} = 39 => (33 - 30)*39 % 95 = 22 => w
- '#' => $\hat{i}$ = 64, a^{-1} = 37 => (64 - 64)*37 % 95 = 0 => a
- G => $\hat{i}$ = 32, a^{-1} = 18 => (32 - 94)*18 % 95 = 24 => y
- E => $\hat{i}$ = 30, a^{-1} = 1 => (30 - 63)*1 % 95 = 62 => !

Сложим всё в одну строку и получим **This is the way!**

Приложение 3. Криптоанализ подстановочных шифров

Часть 1. Шифр простой замены

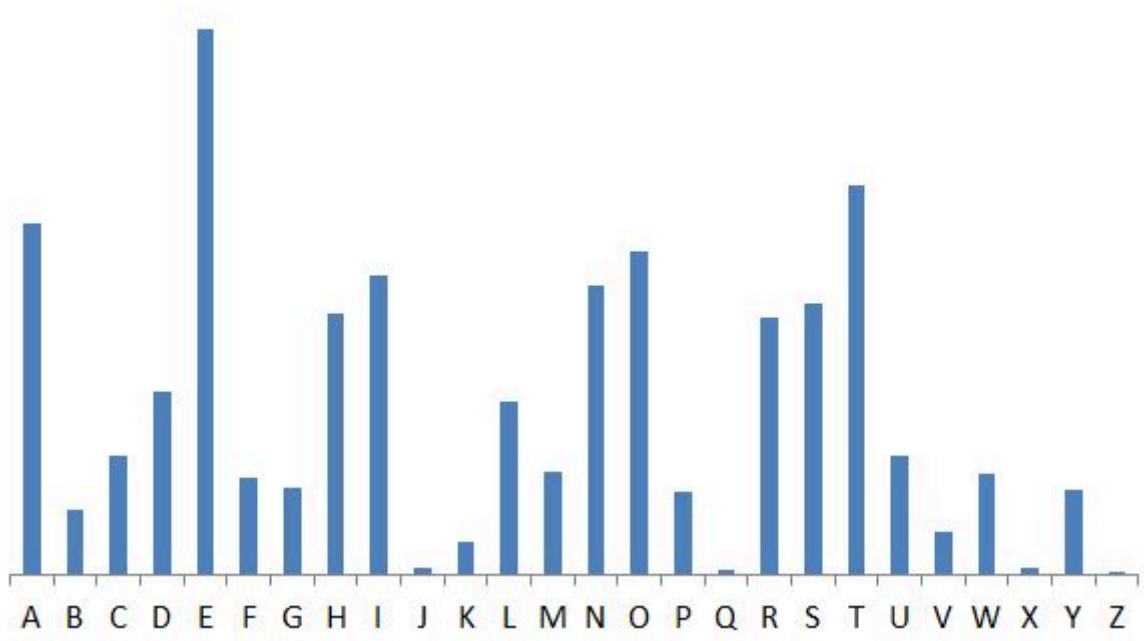
Шифр простой замены поддается частотному анализу так как переносит статистические характеристики языка на шифртекст. Для успешного взлома ключа шифрования шифра простой замены, необходимо иметь достаточно большой текст чтобы корректно проанализировать его.

На рисунке ниже представлена статистика по частотночти букв английского языка в английском тексте.

In [1]:
```
from PIL import Image

Image.open("english.jpg")
```

Out[1]:



In [2]:
```
# Например у нас есть корпус текста который мы хотели бы расшифровать

open_text = """
    Natural language processing is a branch of artificial intelligence
    that enables computers to comprehend, generate, and manipulate human
    language. Natural language processing has the ability to interrogate the
    data with natural language text or voice. This is also called language in.
    Most consumers have probably interacted with NLP without realizing it. For
    instance, NLP is the core technology behind virtual assistants, such as
    the Oracle Digital Assistant, Siri, Cortana, or Alexa. When we ask
    questions of these virtual assistants, NLP is what enables them to not only
    understand the users request, but to also respond in natural language.
    language applies both to written text and speech, and can be applied to all
    human languages. Other examples of tools powered by NLP include web search,
    email spam filtering, automatic translation of text or speech, document
    summarization, sentiment analysis, and grammar spell checking. For example,
    some email programs can automatically suggest an appropriate reply to a
    message based on its content - these programs use language to read, analyze, and
    respond to your message. There are several other terms that are roughly
    synonymous with NLP. Natural language understanding and natural
    language generation refer to using computers to understand and produce
    human language, respectively. NLG has the ability to provide a verbal
    description of what has happened. This is also called language out by
    summarizing by meaningful information into text using a concept known as
    grammar of graphics. In practice, NLU is used to mean NLP. The understanding
    by computers of the structure and meaning of all human languages, allowing
    developers and users to interact with computers using natural sentences and
    communication. Computational linguistics is the scientific field that
    studies computational aspects of human language, while language is the engineering
    discipline concerned with building computational artifacts that understand,
    generate, or manipulate human language. Research on language began shortly after
    the invention of digital computers in the 1950s, and language draws on both
    linguistics and artificial intellect. However, the major breakthroughs of the past
    few years have been powered by machine learning, which is a branch of AI that
    develops systems that learn and generalize from data. Deep learning is a kind of
    machine learning that can learn very complex patterns from large datasets,
    which means that it is ideally suited to learning the complexities of natural
    language from datasets sourced from the web.
"""
```

In [3]:
```
# Немного предобработаю текст - удалю переносы строк и дублирующиеся пробелы
# Также упрощу себе задачу и сведу алфавит до нстрочных букв

open_text = " ".join(open_text.replace("\n", "").split()).lower()
open_text
```

Out[3]:
```
'natural language processing is a branch of artificial intelligence that enables computers to comprehend,
generate, and manipulate human language. natural language processing has the ability to interrogate the da
ta with natural language text or voice. this is also called language in. most consumers have probably inte
racted with nlp without realizing it. for instance, nlp is the core technology behind virtual assistants,
such as the oracle digital assistant, siri, cortana, or alexa. when we ask questions of these virtual assi
stants, nlp is what enables them to not only understand the users request, but to also respond in natural
language. language applies both to written text and speech, and can be applied to all human languages. oth
er examples of tools powered by nlp include web search, email spam filtering, automatic translation of tex
t or speech, document summarization, sentiment analysis, and grammar spell checking. for example, some ema
il programs can automatically suggest an appropriate reply to a message based on its content - these progr
ams use language to read, analyze, and respond to your message. there are several other terms that are rou
ghly synonymous with nlp. natural language understanding and natural language generation refer to using co
mputers to understand and produce human language, respectively. nlg has the ability to provide a verbal de
scription of what has happened. this is also called language out by summarizing by meaningful information
into text using a concept known as grammar of graphics. in practice, nlu is used to mean nlp. the understa
nding by computers of the structure and meaning of all human languages, allowing developers and users to i
nteract with computers using natural sentences and communication. computational linguistics is the scienti
fic field that studies computational aspects of human language, while language is the engineering discipli
ne concerned with building computational artifacts that understand, generate, or manipulate human languag
e. research on language began shortly after the invention of digital computers in the 1950s, and language
draws on both linguistics and artificial intellect. however, the major breakthroughs of the past few years
have been powered by machine learning, which is a branch of ai that develops systems that learn and genera
lize from data. deep learning is a kind of machine learning that can learn very complex patterns from larg
e datasets, which means that it is ideally suited to learning the complexities of natural language from da
tasets sourced from the web.'
```

In [4]:
```
# Зашифрую текст методом простой замены

from simple_cipher import SimpleCipher

encryptor = SimpleCipher()
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[4]:
```
'~DuxJDjdjD~}xD}WdrJz$WHHk~}dkHdDdSJD~$[dzedDJukek$kDjdk~uWjjk}W~$Wdu[DudW~DSjWHd$z%rxuWJHduzd$z%rJW[W~FT
d}W~WJDuWTdD~Fd%D~krxjDuWd[x%D~djD~}xD}W1d~DuxJDjdjD~}xD}WdrJz$WHHk~}d[DHdu[WdDSkjkuyduzdk~uWJJz}DuWdu[WdF
DuDd"ku[d~DuxJDjdjD~}xD}WduWEudzJdtzk$W1du[kHdkHdDjHzd$DjjWFdjD~}xD}Wdk~1d%zHud$z~Hx%WJHd[DtWdrJzSDSjydk~u
WJD$uWFd"ku[d~jrd"ku[zxudJWDjk.k~}dku1dezJdk~HuD~$WTd~jrdkHdu[Wd$zJWduW$[~zjz}ydSW[k~FdtkJuxDjdDHHkHuD~uHT
dHx$[dDHdu[WdzJD$jWdFk}kuDjdDHHkHuD~uTdHkJkTd$zJuD~DTdzJdDjWED1d"[W~d"WdDH{dVxWHukz~Hdzedu[WHWdtkJuxDjdDHH
kHuD~uHTd~jrdkHd"[DudW~DSjWHdu[W%duzd~zudz~jydx~FWJHuD~Fdu[WdxHWJHdJWVxWHuTdSxuduzdDjHzdJWHrz~Fdk~d~DuxJDj
djD~}xD}W1djD~}xD}WdDrrjkWHdSzu[duzd"JkuuW~duWEudD~FdHrWW$[TdD~Fd$D~dSWdDrrjkWFduzdDjjd[x%D~djD~}xD}WH1dzu
[WJdWED%rjWHdzeduzzjHdrz"WJWFdSyd~jrdk~$jxFWd"WSdHWDJ$[TdW%DkjdHrD%dekjuWJk~}TdDxuz%Duk$duJD~HjDukz~dzeduW
EudzJdHrWW$[TdFz$x%W~udHx%%DJk.Dukz~TdHW~uk%W~udD~DjyHkHTdD~Fd}JD%%DJdHrWjjd$[W${k~}1dezJdWED%rjWTdHz%WdW%
DkjdrJz}JD%Hd$D~dDxuz%Duk$DjjydHx}}WHudD~dDrrJzrJkDuWdJWrjyduzdDd%WHHD}WdSDHWFdz~dkuHd$z~uW~udXdu[WHWdrJz}
JD%HdxHWdjD~}xD}WduzdJWDFTdD~Djy.WTdD~FdJWHrz~FduzdyzxJd%WHHD}W1du[WJWdDJWdHWtWJDjdzu[WJduWJ%Hdu[DudDJWdJz
x}[jydHy~z~y%zxHd"ku[d~jr1d~DuxJDjdjD~}xD}Wdx~FWJHuD~Fk~}dD~Fd~DuxJDjdjD~}xD}Wd}W~WJDukz~dJWeWJduzdxHk~}d
$z%rxuWJHduzdx~FWJHuD~FdD~FdrJzFx$Wd[x%D~djD~}xD}WTdJWHrW$uktWjy1d~j}d[DHdu[WdDSkjkuyduzdrJztkFWdDdtWJSDjd
FWH$Jkrukz~dzed"[Dud[DHd[DrrW~WF1du[kHdkHdDjHzd$DjjWFdjD~}xD}WdzxudSydHx%%DJk.k~}dSyd%WD~k~}exjdk~ezJ%Dukz
~dk~uzduWEudxHk~}dDd$z~$Wrud{~z"~dDHd}JD%%DJdzed}JDr[k$H1dk~drJD$uk$WTd~jxdkHdxHWFduzd%WD~d~jr1du[Wdx~FWJH
uD~Fk~}dSyd$z%rxuWJHdzedu[WdHuJx$uxJWdD~Fd%WD~k~}dzedDjjd[x%D~djD~}xD}WHTdDjjz"k~}dFWtWjzrWJHdD~FdxHWJHduz
dk~uWJD$ud"ku[d$z%rxuWJHdxHk~}d~DuxJDjdHW~uW~$WHdD~Fd$z%%x~k$Dukz~1d$z%rxuDukz~Djdjk~}xkHuk$HdkHdu[WdH$kW~
ukek$dekWjFdu[DudHuxFkWHd$z%rxuDukz~DjdDHrW$uHdzed[x%D~djD~}xD}WTd"[kjWdjD~}xD}WdkHdu[WdW~}k~WWJk~}dFkH$kr
jk~Wd$z~$WJ~WFd"ku[dSxkjFk~}d$z%rxuDukz~DjdDJukeD$uHdu[Dudx~FWJHuD~FTd}W~WJDuWTdzJd%D~krxjDuWd[x%D~djD~}x
D}W1dJWHWDJ$[dz~djD~}xD}WdSW}D~dH[zJujydDeuWJdu[Wdk~tW~ukz~dzedFk}kuDjd$z%rxuWJHdk~du[WdCPgRHTdD~FdjD~}xD}
WdFJD"Hdz~dSzu[djk~}xkHuk$HdD~FdDJukek$kDjdk~uWjjW$u1d[z"WtWJTdu[Wd%DLzJdSJWD{u[Jzx}[Hdzedu[WdrDHudeW"dyWD
JHd[DtWdSWW~drz"WJWFdSyd%D$[k~WdjWDJ~k~}Td"[k$[dkHdDdSJD~$[dzedDkdu[DudFWtWjzrHdHyHuW%Hdu[DudjWDJ~dD~Fd}W~
WJDjk.WdeJz%dFDuD1dFWWrdjWDJ~k~}dkHdDd{k~Fdzed%D$[k~WdjWDJ~k~}du[Dud$D~djWDJ~dtWJyd$z%rjWEdrDuuWJ~HdeJz%dj
DJ}WdFDuDHWuHTd"[k$[d%WD~Hdu[DudkudkHdkFWDjjydHxkuWFduzdjWDJ~k~}du[Wd$z%rjWEkukWHdzed~DuxJDjdjD~}xD}WdeJz%
dFDuDHWuHdHzxJ$WFdeJz%du[Wd"WS1'
```

In [5]:
```
# Проанализирую частоту символов в шифртексте

statistic = {}

for i in encrypted_text:
    if statistic.get(i):
        statistic[i] += 1
    else:
        statistic[i] = 1

print(statistic)
```

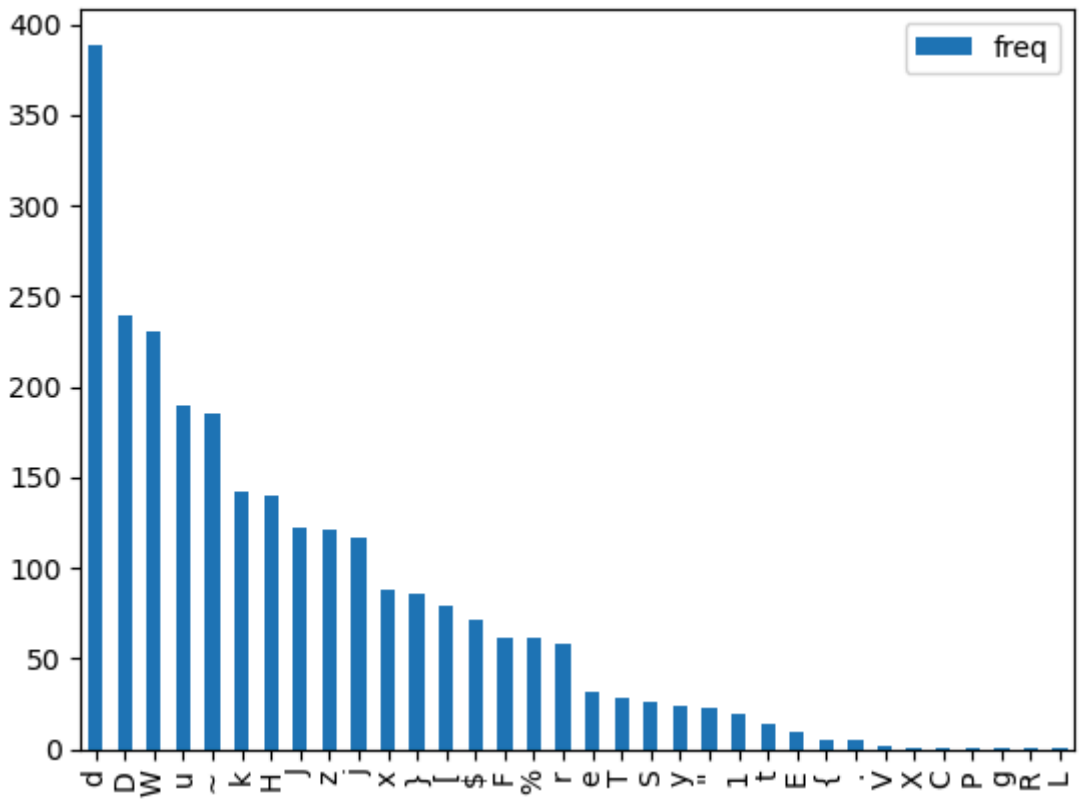
```
{'~': 185, 'D': 239, 'u': 190, 'x': 88, 'J': 122, 'j': 117, 'd': 389, '}': 86, 'W': 230, 'r': 58, 'z': 12
1, '$': 71, 'H': 140, 'k': 142, 'S': 26, '[': 79, 'e': 31, '%': 61, 'F': 61, 'T': 28, '1': 19, 'y': 24,
'"': 23, 'E': 9, 't': 14, '.': 5, '{': 5, 'V': 2, 'X': 1, 'C': 1, 'P': 1, 'g': 1, 'R': 1, 'L': 1}
```

In [6]:
```
# Составлю датафрейм и приведу к табличному виду
# Начертим на графике собранную статистику

import pandas as pd

df = pd.DataFrame.from_dict(statistic, orient='index', columns=['freq']).sort_values('freq', ascending=F
df.plot.bar()
```

Out[6]: <AxesSubplot: >



Частота символа d в шифртексте максимальная и совпадаетс частотой e в английском языке. То же самое можно сказать для следующих пар:

In [7]:
```
# Подбор ключа шифрования (таблицы замен) производится
# Методом восхождения в гору, где сначала подставляются
# элементы максимально приближенные по их частотности,
# и если в результате этого мы получаем текст похожий на
# осмысленный - оставляем кандидата, если нет - проложаем
# подбор
# Среди наиболее вероятных (по частотности) кандидатов
# я сопоставил следующие элементы, которых оказалось
# чтобы понять его суть зашифрованного сообщения, особенно
# если что-то смыслишь в языковых моделях машинного обучения

mapping = {
    'd': ' ', # наиболее распространенный не буквенный символ
    'D': 'a',
    'W': 'e',
    'u': 't',
    '~': 'n',
    'k': 'i',
    'H': 's',
    'J': 'r',
    'z': 'o',
    'j': 'l',
    'x': 'u',
    '}': 'g',
    '[': 'h',
    '$': 'c'
}
```

In [8]:
```
decrypted_array = []

for i in encrypted_text:
    if mapping.get(i):
        decrypted_array.append(mapping[i])
    else:
        decrypted_array.append('*')

result = "".join(decrypted_array)
result
```

Out[8]:
```
'natural language *rocessing is a *ranch o* arti*icial intelligence that ena*les co**uters to co**rehen**
generate* an* *ani*ulate hu*an language* natural language *rocessing has the a*ilit* to interrogate the *a
ta *ith natural language te*t or *oice* this is also calle* language in* *ost consu*ers ha*e *ro*a*l* inte
racte* *ith nl* *ithout reali*ing it* *or instance* nl* is the core technolog* *ehin* *irtual assistants*
such as the oracle *igital assistant* siri* cortana* or ale*a* *hen *e as* *uestions o* these *irtual assi
stants* nl* is *hat ena*les the* to not onl* un*erstan* the users re*uest* *ut to also res*on* in natural
language* language a**lies *oth to *ritten te*t an* s*eech* an* can *e a**lie* to all hu*an languages* oth
er e*a**les o* tools *o*ere* ** nl* inclu*e *e* search* e*ail s*a* *iltering* auto*atic translation o* te*
t or s*eech* *ocu*ent su**ari*ation* senti*ent anal*sis* an* gra**ar s*ell chec*ing* *or e*a**le* so*e e*a
il *rogra*s can auto*aticall* suggest an a**ro*riate re*l* to a *essage *ase* on its content * these *rogr
a*s use language to rea** anal**e* an* res*on* to *our *essage* there are se*eral other ter*s that are rou
ghl* s*non**ous *ith nl** natural language un*erstan*ing an* natural language generation re*er to using co
**uters to un*erstan* an* *ro*uce hu*an language* res*ecti*el** nlg has the a*ilit* to *ro*i*e a *er*al *e
scri*tion o* *hat has ha**ene** this is also calle* language out ** su**ari*ing ** *eaning*ul in*or*ation
into te*t using a conce*t *no*n as gra**ar o* gra*hics* in *ractice* nlu is use* to *ean nl** the un*ersta
n*ing ** co**uters o* the structure an* *eaning o* all hu*an languages* allo*ing *e*elo*ers an* users to i
nteract *ith co**uters using natural sentences an* co**unication* co**utational linguistics is the scienti
*ic *iel* that stu*ies co**utational as*ects o* hu*an language* *hile language is the engineering *isci*li
ne concerne* *ith *uil*ing co**utational arti*acts that un*erstan** generate* or *ani*ulate hu*an language
* research on language *egan shortl* a*ter the in*ention o* *igital co**uters in the ****s* an* language *
ra*s on *oth linguistics an* arti*icial intellect* ho*e*er* the *a*or *rea*throughs o* the *ast *e* *ears
ha*e *een *o*ere* ** *achine learning* *hich is a *ranch o* ai that *e*elo*s s*ste*s that learn an* genera
li*e *ro* *ata* *ee* learning is a *in* o* *achine learning that can learn *er* co**le* *atterns *ro* larg
e *atasets* *hich *eans that it is i*eall* suite* to learning the co**le*ities o* natural language *ro* *a
tasets source* *ro* the *e**'
```

Часть 2. Аффинный шифр

Аффинный шифр по сути ничем от шифра простой замены не отличается. Формула вычисления номера порядкового элемента в аффинном шифре - это по сути линейное выражение некой таблицы замен. Поскольку простая замена не имеет линейной комбинации, а является произвольной комбинацией любых двух элементов из одного словаря - то пространство ключей у шифра простой замены оказжется гораздо шире и равно **n!**, где n - это размерность словаря.

Аффинный шифр так же как и шифр простой замены переносит статистические характеристики языка на шифртекст. Давайте попробуем в этом убедиться и взломать тот же самый текст зашифрованный аффинным шифром с помощью частотного анализа

In [9]:
```
# Зашифрую текст методом аффинного шифрования

from simple_cipher import AffineCipher

encryptor = AffineCipher((8, 80))
encrypted_text = encryptor.encrypt(open_text)
encrypted_text
```

Out[9]:
```
'`=QYA=,+,=`HY=Hr+kAcbrIIX`H+XI+=+_A=`bP+cz+=AQXzXbX=,+X`Qr,,XHr`br+QP=Q+r`=_,rI+bc>kYQrAI+Qc+bc>kArPr`j +
Hr`rA=Qr +=`j+>=`XkY,=Qr+PY>=`+,=`HY=Hrp+`=QYA=,+,=`HY=Hr+kAcbrIIX`H+P=I+QPr+=_X,XQ?+Qc+X`QrAAcH=Qr+QPr+j=
Q=+%XQP+`=QYA=,+,=`HY=Hr+Qr-Q+cA+6cXbrp+QPXI+XI+=,Ic+b=,,rj+,=`HY=Hr+X`p+>cIQ+bc`IY>rAI+P=6r+kAc_=_,?+X`Qr
A=bQrj+%XQP+`,k+%XQPcYQ+Ar=,X{X`H+XQp+zcA+X`IQ=`br +`,k+XI+QPr+bcAr+QrbP`c,cH?+_rPX`j+6XAQY=,+=IIXIQ=`QI +
IYbP+=I+QPr+cA=b,r+jXHXQ=,+=IIXIQ=`Q +IXAX +bcAQ=`= +cA+=,r-=p+%Pr`+%r+=I$+sYrIQXc`I+cz+QPrIr+6XAQY=,+=IIX
IQ=`QI +`,k+XI+%P=Q+r`=_,rI+QPr>+Qc+`cQ+c`,?+Y`jrAIQ=`j+QPr+YIrAI+ArsYrIQ +_YQ+Qc+=,Ic+ArIkc`j+X`+`=QYA=,
+,=`HY=Hrp+,=`HY=Hr+=kk,XrI+_cQP+Qc+%AXQQr`+Qr-Q+=`j+IkrrbP +=`j+b=`+_r+=kk,Xrj+Qc+=,,+PY>=`+,=`HY=HrIp+cQ
PrA+r-=>k,rI+cz+Qcc,I+kc%rArj+_?+`,k+X`b,Yjr+%r_+Ir=AbP +r>=X,+Ik=>+zX,QrAX`H +=YQc>=QXb+QA=`I,=QXc`+cz+Qr
-Q+cA+IkrrbP +jcbY>r`Q+IY>>=AX{=QXc` +Ir`QX>r`Q+=`=,?IXI +=`j+HA=>>=A+Ikr,,+bPrb$X`Hp+zcA+r-=>k,r +Ic>r+r>
=X,+kAcHA=>I+b=`+=YQc>=QXb=,,?+IYHHrIQ+=`+=kkAckAX=Qr+Ark,?+Qc+=+>rII=Hr+_=Irj+c`+XQI+bc`Qr`Q+h+QPrIr+kAcH
A=>I+YIr+,=`HY=Hr+Qc+Ar=j +=`=,?{r +=`j+ArIkc`j+Qc+?cYA+>rII=Hrp+QPrAr+=Ar+Ir6rA=,+cQPrA+QrA>I+QP=Q+=Ar+Ac
YHP,?+I?`c`?>cYI+%XQP+`,kp+`=QYA=,+,=`HY=Hr+Y`jrAIQ=`jX`H+=`j+`=QYA=,+,=`HY=Hr+Hr`rA=QXc`+ArzrA+Qc+YIX`H+b
c>kYQrAI+Qc+Y`jrAIQ=`j+=`j+kAcjYbr+PY>=`+,=`HY=Hr +ArIkrbQX6r,?p+`,H+P=I+QPr+=_X,XQ?+Qc+kAc6Xjr+=+6rA_=,+j
rIbAXkQXc`+cz+%P=Q+P=I+P=kkr`rjp+QPXI+XI+=,Ic+b=,,rj+,=`HY=Hr+cYQ+_?+IY>>=AX{X`H+_?+>r=`X`HzY,+X`zcA>=QXc`
+X`Qc+Qr-Q+YIX`H+=+bc`brkQ+$`c%`+=I+HA=>>=A+cz+HA=kPXbIp+X`+kA=bQXbr +`,Y+XI+YIrj+Qc+>r=`+`,kp+QPr+Y`jrAIQ
=`jX`H+_?+bc>kYQrAI+cz+QPr+IQAYbQYAr+=`j+>r=`X`H+cz+=,,+PY>=`+,=`HY=HrI +=,,c%X`H+jr6r,ckrAI+=`j+YIrAI+Qc+
X`QrA=bQ+%XQP+bc>kYQrAI+YIX`H+`=QYA=,+Ir`Qr`brI+=`j+bc>>Y`Xb=QXc`p+bc>kYQ=QXc`=,+,X`HYXIQXbI+XI+QPr+IbXr`Q
XzXb+zXr,j+QP=Q+IQYjXrI+bc>kYQ=QXc`=,+=IkrbQI+cz+PY>=`+,=`HY=Hr +%PX,r+,=`HY=Hr+XI+QPr+r`HX`rrAX`H+jXIbXk,
X`r+bc`brA`rj+%XQP+_YX,jX`H+bc>kYQ=QXc`=,+=AQXz=bQI+QP=Q+Y`jrAIQ=`j +Hr`rA=Qr +cA+>=`XkY,=Qr+PY>=`+,=`HY=H
rp+ArIr=AbP+c`+,=`HY=Hr+_rH=`+IPcAQ,?+=zQrA+QPr+X`6r`QXc`+cz+jXHXQ=,+bc>kYQrAI+X`+QPr+D~9vI +=`j+,=`HY=Hr+
jA=%I+c`+_cQP+,X`HYXIQXbI+=`j+=AQXzXbX=,+X`Qr,,rbQp+Pc%r6rA +QPr+>=5cA+_Ar=$QPAcYHPI+cz+QPr+k=IQ+zr%+?r=AI
+P=6r+_rr`+kc%rArj+_?+>=bPX`r+,r=A`X`H +%PXbP+XI+=+_A=`bP+cz+=X+QP=Q+jr6r,ckI+I?IQr>I+QP=Q+,r=A`+=`j+Hr`rA
=,X{r+zAc>+j=Q=p+jrrk+,r=A`X`H+XI+=+$X`j+cz+>=bPX`r+,r=A`X`H+QP=Q+b=`+,r=A`+6rA?+bc>k,r-+k=QQrA`I+zAc>+,=A
Hr+j=Q=IrQI +%PXbP+>r=`I+QP=Q+XQ+XI+Xjr=,,?+IYXQrj+Qc+,r=A`X`H+QPr+bc>k,r-XQXrI+cz+`=QYA=,+,=`HY=Hr+zAc>+j
=Q=IrQI+IcYAbrj+zAc>+QPr+%r_p'
```

In [10]:
```
# Проанализирую частоту символов в шифртексте

statistic = {}

for i in encrypted_text:
    if statistic.get(i):
        statistic[i] += 1
    else:
        statistic[i] = 1

print(statistic)
```

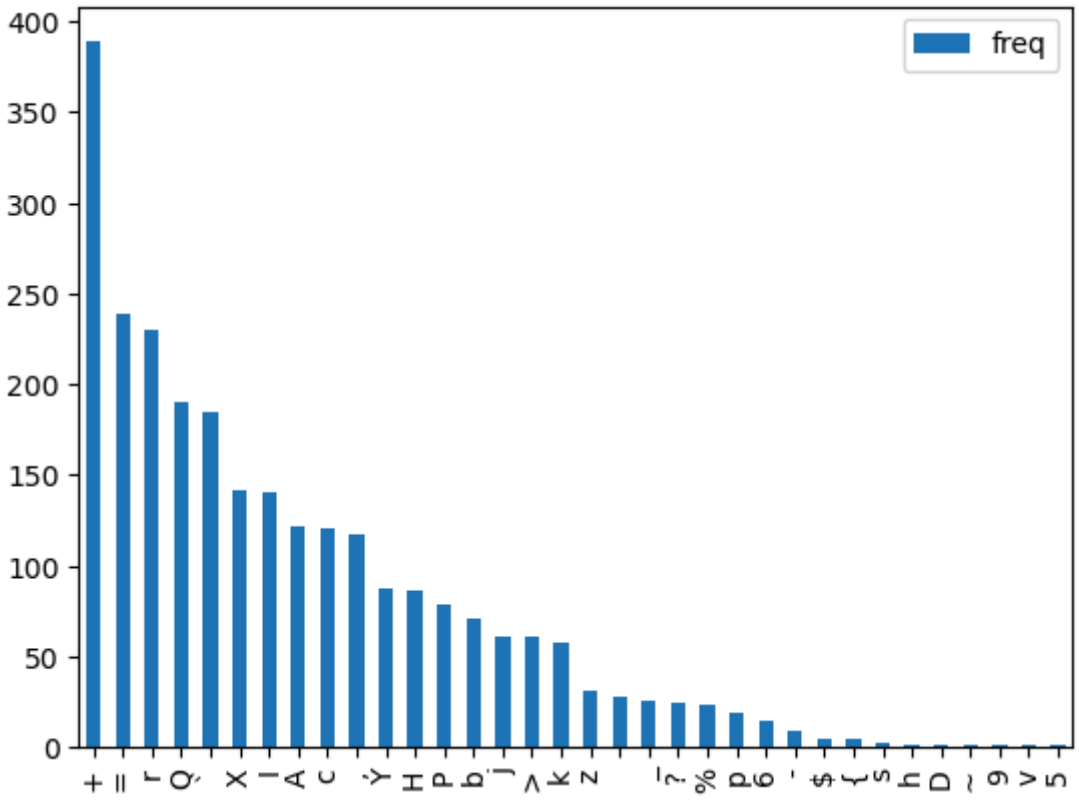
```
{'`': 185, '=': 239, 'Q': 190, 'Y': 88, 'A': 122, ',': 117, '+': 389, 'H': 86, 'r': 230, 'k': 58, 'c': 12
1, 'b': 71, 'I': 140, 'X': 142, '_': 26, 'P': 79, 'z': 31, '>': 61, 'j': 61, ' ': 28, 'p': 19, '?': 24,
'%': 23, '-': 9, '6': 14, '{': 5, '$': 5, 's': 2, 'h': 1, 'D': 1, '~': 1, '9': 1, 'v': 1, '5': 1}
```

In [11]:
```
# Составлю датафрейм и приведу к табличному виду
# Начертим на графике собранную статистику

import pandas as pd

df = pd.DataFrame.from_dict(statistic, orient='index', columns=['freq']).sort_values('freq', ascending=F
df.plot.bar()
```

Out[11]: <AxesSubplot: >



In [12]:
```
# Методом пристального взгляда и простого перебора
# кандидатов исходя из таблицы частотности
# попробуем взломать и аффинный метод шифрования

mapping = {
    '+': ' ', # наиболее распространенный не буквенный символ
    '=': 'a',
    'r': 'e',
    'Q': 't',
    '`': 'n',
    'X': 'i',
    'I': 's',
    'A': 'r',
    'c': 'o',
    ',': 'l',
    'Y': 'u',
    'H': 'g',
    'P': 'h',
    'b': 'c'
}
```

In [13]:
```
decrypted_array = []

for i in encrypted_text:
    if mapping.get(i):
        decrypted_array.append(mapping[i])
    else:
        decrypted_array.append('*')

result = "".join(decrypted_array)
result
```

Out[13]:
```
'natural language *rocessing is a *ranch o* arti*icial intelligence that ena*les co**uters to co**rehen**
generate* an* *ani*ulate hu*an language* natural language *rocessing has the a*ilit* to interrogate the *a
ta *ith natural language te*t or *oice* this is also calle* language in* *ost consu*ers ha*e *ro*a*l* inte
racte* *ith nl* *ithout reali*ing it* *or instance* nl* is the core technolog* *ehin* *irtual assistants*
such as the oracle *igital assistant* siri* cortana* or ale*a* *hen *e as* *uestions o* these *irtual assi
stants* nl* is *hat ena*les the* to not onl* un*erstan* the users re*uest* *ut to also res*on* in natural
language* language a**lies *oth to *ritten te*t an* s*eech* an* can *e a**lie* to all hu*an languages* oth
er e*a**les o* tools *o*ere* ** nl* inclu*e *e* search* e*ail s*a* *iltering* auto*atic translation o* te*
t or s*eech* *ocu*ent su**ari*ation* senti*ent anal*sis* an* gra**ar s*ell chec*ing* *or e*a**le* so*e e*a
il *rogra*s can auto*aticall* suggest an a**ro*riate re*l* to a *essage *ase* on its content * these *rogr
a*s use language to rea** anal**e* an* res*on* to *our *essage* there are se*eral other ter*s that are rou
ghl* s*non**ous *ith nl** natural language un*erstan*ing an* natural language generation re*er to using co
**uters to un*erstan* an* *ro*uce hu*an language* res*ecti*el** nlg has the a*ilit* to *ro*i*e a *er*al *e
scri*tion o* *hat has ha**ene** this is also calle* language out ** su**ari*ing ** *eaning*ul in*or*ation
into te*t using a conce*t *no*n as gra**ar o* gra*hics* in *ractice* nlu is use* to *ean nl** the un*ersta
n*ing ** co**uters o* the structure an* *eaning o* all hu*an languages* allo*ing *e*elo*ers an* users to i
nteract *ith co**uters using natural sentences an* co**unication* co**utational linguistics is the scienti
*ic *iel* that stu*ies co**utational as*ects o* hu*an language* *hile language is the engineering *isci*li
ne concerne* *ith *uil*ing co**utational arti*acts that un*erstan** generate* or *ani*ulate hu*an language
* research on language *egan shortl* a*ter the in*ention o* *igital co**uters in the ****s* an* language *
ra*s on *oth linguistics an* arti*icial intellect* ho*e*er* the *a*or *rea*throughs o* the *ast *e* *ears
ha*e *een *o*ere* ** *achine learning* *hich is a *ranch o* ai that *e*elo*s s*ste*s that learn an* genera
li*e *ro* *ata* *ee* learning is a *in* o* *achine learning that can learn *er* co**le* *atterns *ro* larg
e *atasets* *hich *eans that it is i*eall* suite* to learning the co**le*ities o* natural language *ro* *a
tasets source* *ro* the *e**'
```

Часть 3. Аффинный рекуррентный шифр

Поскольку аффинный рекуррентный шифр на каждом шаге вырабатывает новые ключи для шифрования - то один и тот же символ может быть зашифрован разными символами, и наоборот, два разных шифра могут быть зашифрованы одинаковыми символами.

Таким образом частотный анализ не принесет успеха в криптоанализе. Однако поскольку пространство ключей не столь велико и зависит по-сути от величины словаря, то данный шифр может быть взломан методом грубой силы - то есть простым перебором двух возможных вариантов ключей и при много кратном расшифровании зашифрованного текста - получить в итоге исходный текст.

Спасибо за внимание!