МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ПРЕДМЕТУ

«Программирование криптографических алгоритмов»

Выполнил:

Барышников С.С.
гр. 191-351

Преподаватель:

Бутакова Н.Г.

Москва 2021 г.

Содержание

Аннотация 3
Постоянный модуль 4
Блок A: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ 5
1. Шифр простой замены АТБАШ 5
2. ШИФР ЦЕЗАРЯ 7
3. Квадрат Полибия 9
Блок B: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ 12
4. Шифр Тритемия 12
5. Шифр Белазо 15
Блок C: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ 18
8. Матричный шифр 18
9. Шифр Плейфера 21
D: ШИФРЫ ПЕРЕСТАНОВКИ 26
10. Шифр вертикальной перестановки 26
11. Решетка Кардано 29
E: ШИФРЫ ГАММИРОВАНИЯ 33
13. Одноразовый блокнот К.Шеннона 33
14. Гаммирование ГОСТ 28147-89 38
F: ПОТОЧНЫЕ ШИФРЫ 43
15. A5 /1 43

Аннотация

Среда программирования: Visual Studio Code

Язык программирования: Python 3

Процедуры для запуска программы: $ python3 <имя_файла>.py

Пословица-тест: Время, приливы и отливы не ждут человека.

Текст для проверки работы: Вот пример статьи на тысячу символов. Это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. В таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. Но можно и без него. На тысячу символов рекомендовано использовать один или два ключа и одну картину. Текст на тысячу символов это сколько примерно слов? Статистика показывает, что тысяча включает в себя стопятьдесят или двести слов средней величины. Но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. В копирайтерской деятельности принято считать тысячи с пробелами или без. Учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. Считать пробелы заказчики не любят, так как это пустое место. Однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. Согласитесь, читать слитный текст без единого пропуска, никто не будет. Но большинству нужна цена за тысячу знаков без пробелов.

Интерфейс: #в разработке#

Постоянный модуль

Код модуля base.py используемый для предотвращения дублирования кода, используется во всех последующих программах:

```
import re

alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

dict = {'.': 'тчк', ',': 'зпт'}


def replace_all_to(input_text, dict):
    input_text = input_text.replace(' ', '')
    for i, j in dict.items():
        input_text = input_text.replace(i, j)
    return input_text


def replace_all_from(input_text, dict):
    for i, j in dict.items():
        input_text = input_text.replace(j, i)
    return input_text


def file_to_string(name):
    with open(name) as f:
        input_short_text = " ".join([l.rstrip() for l in f]) + ' '
    return input_short_text.lower()


def input_for_cipher_short():
    return replace_all_to(file_to_string('short.txt'), dict)


def input_for_cipher_long():
    return replace_all_to(file_to_string('long.txt'), dict)


def output_from_decrypted(decrypted_text):
    return replace_all_from(decrypted_text, dict)
```

Блок А: ШИФРЫ ОДНОЗНАЧНОЙ ЗАМЕНЫ

1. Шифр простой замены АТБАШ

Атбаш — простой шифр подстановки для алфавитного письма. Правило шифрования состоит в замене **i**-й буквы алфавита буквой с номером **n-i+1**, где **n** — число букв в алфавите.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

# функция преобразования текста
def atbash(input):
    return input.translate(str.maketrans(
        alphabet + alphabet.upper(), alphabet[::-1] + alphabet.upper()[::-
1]))


# вывод результатов работы программы
print(f'''
ШИФР АТБАШ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{atbash(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(atbash(atbash(input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{atbash(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(atbash(atbash(input_for_cipher_long())))}
''')
```

Тестирование:

```
root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01_1_atbash.py
ШИФР АТБАШ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
эоъта чпм поцуцэд ц рмуцэд съ шылм зъурэъфя мзф

Расшифрованный текст:
время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
```

эрм поцтъо нмямгц ся мдназл нцтэрурэ мзф вмр ырнмямрзср тяуъсгфцх мъфнм чпм рпмцтяугср прыйрыаёцх ыуа фяомрзъф мрэяорэ э цсмъосъм цуц тяьячцсяй цуц ыуа съюругжцй цскротяицрссдй плюуцфяицх мзф э мяфрт мъфнмъ оъыфр юдэяъм юруъъ ыэлй цуц мощй яючяиъэ ц рюдзср рыцс прычяьрурэрф мзф ср тршср ц юъч съър мзф ся мдназл нцтэрурэ оъфртъсырэяср цнпругчрэямг рыцс цуц ыэя фубзя ц рысл фяомцсл мзф мъфнм ся мдназл нцтэрурэ вмр нфругфр поцтъоср нурэ эпо нмямцнмцфя прфячдэяъм чпм змр мдназя эфубзяъм э нъюа нмрпамгыънам цуц ыэънмц нурэ ноъысъх эъуцзцсд мзф ср чпм ънуц чурлпрмоъюуамг поъыурьятц чпм нрбчятц ц ыольцтц зянматц оъзц ся рыцс цуц ыэя нцтэруя чпм мр фруцзънмэр нурэ съцчтъсср эрчоянмяъм мзф э фрпцояхмъонфрх ыъамъугсрнмц поцсамр нзцмямг мдназц н порюъуятц цуц юъч мзф лзъм порюъурэ лэъуцзцэяъм рюеът мъфнмя поцтъоср ся нмр цуц ыэънмц нцтэрурэ цтъсср нмругфр ояч тд оячыъуаът нурэя нэрюрысдт порнмояснмэрт мзф нзцмямг порюъуд чяфячзцфц съ убюам чпм мяф фяф вмр плнмръ тънмр мзф рысяфр съфрмродъ кцотд ц юцошц эцыам нпояэъыуцэдт нмяэцмг нмрцтрнмг чя мдназл нцтэрурэ н порюъуятц чпм нзцмяа прнуъысцъ эяшсдт вуътъсмрт фязънмэъссрьр эрнпоцамца мзф нрьуянцмънг чпм зцмямг нуцмсдх мъфнм юъч ъыцсрьр порплнфя чпм сцфмр съ юлыъм мзф ср юругжцснмэл слшся иъся чя мдназл чсяфрэ юъч порюъурэ мзф

Расшифрованный текст:
вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но можно и без него. на тысячу символов рекомендовано использовать один или два ключа и одну картину. текст на тысячу символов это сколько примерно слов? статистика показывает, что тысяча включает в себя стопятьдесят или двести слов средней величины. но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. в копирайтерской деятельности принято считать тысячи с пробелами или без. учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это пустое место. однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна цена за тысячу знаков без пробелов.

2. ШИФР ЦЕЗАРЯ

Шифр Цезаря, также известный как шифр сдвига, код Цезаря или сдвиг Цезаря — один из самых простых и наиболее широко известных методов шифрования.

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import random

key = int(input('Введите ключ: '))

# функция шифровки
def caesar_encode(input, step):
    return input.translate(
        str.maketrans(alphabet, alphabet[step:] + alphabet[:step]))

# функция дешифровки
def caesar_decode(input, step):
    return input.translate(
        str.maketrans(alphabet[step:] + alphabet[:step], alphabet))
# вывод результатов работы программы
print(f'''
ШИФР ЦЕЗАРЯ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{caesar_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(caesar_decode(caesar_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{caesar_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(caesar_decode(caesar_encode(
    input_for_cipher_long(), key), key))}
''')
```

Тестирование:

```
root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01_2_caesar.py
```

ШИФР ЦЕЗАРЯ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
бпдлю жос опзкзбъ з нскзбъ мд ёгтс цдкнбдйя сцй

Расшифрованный текст:
время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
бнс опзлдп рсясыз мя сърюцт рзлбнкнб сцй ьсн гнрсяснцмн лякдмыйзи сдйрс жос носзлякымн онгфнгюшзи гкю йяпснцдй снбяпнб б змсдпмдс зкз лявяжзмяф зкз гкю мданкычзф змунпляхзнммъф отакзйяхзи сцй б сяйнл сдйрсд пдгйн аъбядс анкдд гбтф зкз спеф яажяхдб з наъцмн нгзм онгжявнкнбнй сцй мн лнёмн з адж мдвн сцй мя сърюцт рзлбнкнб пдйнлдмгнбямн зронкыжнбясы нгзм зкз гбя йкэця з нгмт йяпсзмт сцй сдйрс мя сърюцт рзлбнкнб ьсн рйнкыйн опзлдпмн ркнб боп рсясзрсзйя онйяжъбядс жос цсн сърюця бйкэцядс б рдаю рсн оюсыгдрюс зкз гбдрсз ркнб рпдгмди бдкзцзмъ сцй мн жос дркз жкнтонспдакюсы опдгкнвялз жос рнэжялз з гптвзлз цярсюлз пдцз мя нгзм зкз гбя рзлбнкя жос сн йнкзцдрсбн ркнб мдзжлдммн бнжпярсядс сцй б йнозпяисдпрйни гдюсдкымнрсз опзмюсн рцзсясы сърюцз р опнадкялз зкз адж сцй тцдс опнадкнб тбдкзцзбядс нащдл сдйрся опзлдпмн мя рсн зкз гбдрсз рзлбнкнб злдммн рснкыйн пяж лъ пяжгдкюдл ркнбя рбнангмъл опнрспямрсбнл сцй рцзсясы опнадкъ жяйяжцзйз мд кэаюс жос сяй йяй ьсн отрснд лдрсн сцй нгмяйн мдйнснпъд узплъ з азпёз бзгюс ропябдгкзбъл рсябзсы рснзлнрсы жя сърюцт рзлбнкнб р опнадкялз жос рцзсяю онркдгмзд бяёмъл ькдлдмснл йяцдрсбдммнвн бнропзюсзю сцй рнвкярзсдры жос цзсясы ркзсмъи сдйрс адж дгзмнвн опнотрйя жос мзйсн мд атгдс сцй мн анкычзмрсбт мтёмя хдмя жя сърюцт жмяйнб адж опнадкнб сцй

Расшифрованный текст:
вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но можно и без него. на тысячу символов рекомендовано использовать один или два ключа и одну картину. текст на тысячу символов это сколько примерно слов? статистика показывает, что тысяча включает в себя сто пятьдесят или двести слов средней величины. но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. в копирайтерской деятельности принято считать тысячи с пробелами или без. учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это пустое место. однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна цена за тысячу знаков без пробелов.

3. Квадрат Полибия

Квадрат Полибия – метод шифрования текстовых данных с помощью замены символов, впервые предложен греческим историком и полководцем Полибием.

К каждому языку отдельно составляется таблица шифрования с одинаковым (не обязательно) количеством пронумерованных строк и столбцов, параметры которой зависят от его мощности (количества букв в алфавите). Берутся два целых числа, произведение которых ближе всего к количеству букв в языке — получаем нужное число строк и столбцов. Затем вписываем в таблицу все буквы алфавита подряд — по одной на каждую клетку. При нехватке клеток можно вписать в одну две буквы (редко употребляющиеся или схожие по употреблению).

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

# квадрат полибия
hard_dictionary = {"а": "11", "б": "12", "в": "13",
                   "г": "14", "д": "15", "е": "16", "ё": "21",
                   "ж": "22", "з": "23", "и": "24", "й": "25",
                   "к": "26", "л": "31", "м": "32", "н": "33",
                   "о": "34", "п": "35", "р": "36", "с": "41",
                   "т": "42", "у": "43", "ф": "44", "х": "45",
                   "ц": "46", "ч": "51", "ш": "52", "щ": "53",
                   "ъ": "54", "ы": "55", "ь": "56", "э": "61",
                   "ю": "62", "я": "63"}
# функция шифровки
def square_encode(input):
    new_txt = ""
    for x in input:
        if x in hard_dictionary:
            new_txt += hard_dictionary.get(x)
        else:
            new_txt += (x + x)
    return new_txt
# функция дешифровки
def square_decode(input):
    new_txt = ""
    list_fraze = []
    step = 2
    for i in range(0, len(input), 2):
        list_fraze.append(input[i:step])
        step += 2
    key_hard_dictionary_list = list(hard_dictionary.keys())
    val_hard_dictionary_list = list(hard_dictionary.values())
```

```
    for x in list_fraze:
        if x in val_hard_dictionary_list:
            i = val_hard_dictionary_list.index(x)
            new_txt += key_hard_dictionary_list[i]
        else:
            new_txt += x[0:1]
    return new_txt
# вывод результатов работы программы
print(f'''
КВАДРАТ ПОЛИБИЯ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{square_encode(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(square_decode(square_encode(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{square_encode(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(square_decode(square_encode(
    input_for_cipher_long())))}
''')
```

Тестирование:

```
root@DESKTOP-05UI9FD:~/crypt# /bin/python3 /root/crypt/lab01_3_square.py

КВАДРАТ ПОЛИБИЯ:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
1336163263  233542  35362431241355  24  344231241355  3316  22154342
5116313413162611  425126

Расшифрованный текст:
время, приливы и отливы не ждут человека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
133442  353624321636  414211425624  3311  425541635143  4124321334313413
425126  614234  15344142114234513334  321131163356262425  4216264142  233542
343542243211315633334  353415453415635324 25  153163  2611364234511626
42341311363413  13  2433421636331642  243124  321114112324331145  243124
153163  331612343156522445  24334434363211462434333335545
354312312426114624 25  425126  13  4211263432  421626414216  3616152634
125513111642  1234311616  15134345  243124  42362145  11122311461613  24
```

```
341255513334  34152433  353415231114343134133426  425126  3334  3234223334
24  121623  33161434  425126  3311  425541635143  4124321334313413
36162634321633153413113334  24413534315623341311 4256  34152433  243124
151311  2631625111  24  34153343  26113642243343  425126  4216264142  3311
425541635143  4124321334313413  614234  41263431562634  3536243216363334
41313413  133536  41421142244142242611  35342611235513111642  233542  514234
425541635111  1326316251111642  13  41161263  414234  356342561516416342
243124  151316414224  41313413  41361615331625  1316312451243355  425126
3334  233542  16413124  233134433534423616123163425 6  353616153134141132 24
233542  41346223113224  24  15364314243224  51114142633224  36165124  3311
34152433  243124  151311  41243213343111  233542  4234  263431245116414213 34
41313413  331624233216333334  133423361141421116 42  425126  13
263435243611254216364126342 5  151663421631563334414224  353624336342 34
41512442114256  425541635124  41  353634121631113224  243124  121623  425126
43511642  3536341216313413  43131631245124131116 42  3412541632  421626414211
3536243216363334  3311  414234  243124  151316414224  4124321334313413
243216333334  41423431562634  361123  3255  361123151631631632  4131341311
411334123415335532  353634414236113341421334 32  425126  41512442114256
353634121631 55  231126112351242624  3316  3162126342  233542  421126  261126
614234  354341423416  3216414234  425126  341533112634  331626344234365516
4424363255  24  1224362224  1324156342  413536111316153124135532
41421113244256  414234243234414256  2311  425541635143  4124321334313413  41
353634121631113224  233542  415124421163  353441311615332416  131122335532
613116321633423432  261151164142131633333414 34  133441353624634224 63  425126
413414311141244216415 6  233542  512442114256  41312442335525  4216264142
121623  16152433341434  3536343543412611  233542  3324264234  3316
1243151642  425126  3334  12343156522433414213 43  3343223311  46163311  2311
425541635143  233311263413  121623  3536341216313413  425126
```

Расшифрованный текст:

вот пример статьи на тысячу символов. это достаточно маленький текст, оптимально подходящий для карточек товаров в интернет или магазинах или для небольших информационных публикаций. в таком тексте редко бывает более двух или трёх абзацев и обычно один подзаголовок. но можно и без него. на тысячу символов рекомендовано использовать один или два ключа и одну картину. текст на тысячу символов это сколько примерно слов? статистика показывает, что тысяча включает в себя сто пятьдесят или двести слов средней величины. но, если злоупотреблять предлогами, союзами и другими частями речи на один или два символа, то количество слов неизменно возрастает. в копирайтерской деятельности принято считать тысячи с пробелами или без. учет пробелов увеличивает объем текста примерно на сто или двести символов именно столько раз мы разделяем слова свободным пространством. считать пробелы заказчики не любят, так как это пустое место. однако некоторые фирмы и биржи видят справедливым ставить стоимость за тысячу символов с пробелами, считая последние важным элементом качественного восприятия. согласитесь, читать слитный текст без единого пропуска, никто не будет. но большинству нужна цена за тысячу знаков без пробелов.

Блок В: ШИФРЫ МНОГОЗНАЧНОЙ ЗАМЕНЫ

4. Шифр Тритемия

Шифр Тритемия предполагал использование алфавитной таблицы. Он использовал эту таблицу для многоалфавитного зашифрования самым простым из возможных способов: первая буква текста шифруется первым алфавитом, вторая буква — вторым и т. д. В этой таблице не было отдельного алфавита открытого текста, для этой цели служил алфавит первой строки. Таким образом, открытый текст, начинающийся со слов HUNC CAVETO VIRUM ..., приобретал вид HXPF GFBMCZ FUEIB ... .

Преимущество этого метода шифрования по сравнению с методом Альберти состоит в том, что с каждой буквой задействуется новый алфавит. Альберти менял алфавиты лишь по-сле трех или четырех слов. Поэтому его шифртекст состоял из отрезков, каждый из которых обладал закономерностями открытого текста, которые помогали вскрыть криптограмму. Побуквенное зашифрование не дает такого преимущества. Шифр Тритемия является также первым нетривиальным примером периодического шифра. Так называется многоалфавитный шифр, правило зашифрования которого состоит в использовании периодически повторяющейся последовательности простых замен.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

# функция шифровки
def trithemius_decode(input):
    decode: str = ""
    k = 0
    for position, symbol in enumerate(input):
        index = (alphabet.find(symbol) + k) % len(alphabet)
        decode += alphabet[index]
        k -= 1
    return decode

# функция дешифровки
def trithemius_encode(input):
    encode = ""
    k = 0
    for position, symbol in enumerate(input):
        index = (alphabet.find(symbol) + k) % len(alphabet)
        encode += alphabet[index]
        k += 1
    return encode
```

```
# вывод результатов работы программы
print(f'''
Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{trithemius_encode(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(trithemius_decode(trithemius_encode(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{trithemius_encode(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(trithemius_decode(trithemius_encode(
    input_for_cipher_long())))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-
1/lab02_4_trithemius.py

Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
всжпгмхщчщтцфоичюгэыхпгыюьммтбимбелвхып

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
впфтфнтлшъькюицьпгмдтлизаеыижкярцкюфсзучщзышвщыъоыхяоюэяиьгкмглпмотйогпбухччн
аърмзшъхютяхжйжряёолаярпдемтшлшцоэфшцыпусъьвхладвюжыкгаомюымъофьчъчлгцээюмзгз
цвагщрдёпхйвувнтсшлтъьпьссщюсмфущзчёдюяяюрузлйфуйъёзпиапнхьпкзябвшюджжэвыялйн
впмхыпухфчршъхоучюцхвжмбешлхмыфсриндспузчмушчръсэсрябьёеегфбиэьпъндйплпнйизух
игмцэуюеэуезяеллсёовиртовхяцеюыътчныщэсндбеядвугзйлейгпнпуотжешъиьуэяцщпааэуь
ршчэлвкофрнтьувыезсужбкряпафсрдёгехйфэямыпжкиедзхчошучлььссьфъучяэмяеулсёйлёо
тёуоммсхышшэьсоъогнвдщщвёыщждммррояйгрнокшьмушхеобгряьеънаёщияекжкиедгхнтицфй
тыяаэъяэыкщжищжкёйюкгззнжрсузхпщйьюмтбвзьфюгеязшгмамоинйежвцйсгхчыэфьючэпдбел
юичкхмцньхзсртсейсжфстцфнтцвзьёкщзёзжъяжкясбемкъмжёлъчкерщауъвдтгеюгьижиьорн
имкжёйчщрътчныщэсщючбвдзйжецёкнюжмьтгликтжнцьчыпобтаувшгсдзйимонюмсудсрчэсщат
эляйюаятчяпэцвшбсджёлдлягвцмхщъифхлчкбюаедёгъгьйлинузмгнбмссйрхъчъёрбцяьлъуье
ьящбэщннмоявёёомжбущыймфяйавяъявёзтяшлхмыфсриндспузшчщшлршнышшбеелюиччиимзввм
иёдгктбуефочръаэыысйшёзшьёжиюкямрпсквцнещяуыщошнашгцдпьеиоблъншойзтоэмцйршъйо
тррцьюуавдгейигщвкокжйтппзешйлъыбхщыоэымйыбёзшкёмбиьугмаивхяцеюырчкцыппфшбгвх
ъвёълсё
```

```
Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернословстатистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименносто
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

5. Шифр Белазо

В 1553 Джованни Баттиста Белазо предложил использовать для многоалфавитного шифра буквенный, легко запо-минаемый ключ, который он назвал паролем. Паролем могло служить слово или фраза. Пароль периодически записывался над открытым текстом. Буква пароля, расположенная над буквой текста, указывала на алфавит таблицы, который исполь-зовался для зашифрования этой буквы. Например, это мог быть алфавит из таблицы Тритемия, первой буквой которого являлась буква пароля. Однако Белазо, как и Тритемий, использовал в качестве алфавитов шифра обычные алфавиты.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

key = str(input('Введите ключ: '))

# функция шифровки
def bellaso_decode(input, key):
    decrypted = ''
    offset = 0
    for ix in range(len(input)):
        if input[ix] not in alphabet:
            output = input[ix]
            offset += -1
        elif (alphabet.find(input[ix])) > (len(alphabet) - (alphabet.find(key
[((ix + offset) % len(key))]))) - 1):
            output = alphabet[(alphabet.find(
                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))])))
) % 33]
        else:
            output = alphabet[alphabet.find(
                input[ix]) - (alphabet.find(key[((ix + offset) % len(key))])))
]
        decrypted += output
    return decrypted

# функция дешифровки
def bellaso_encode(input, key):
    encoded = ''
    offset = 0
    for ix in range(len(input)):
        if input[ix] not in alphabet:
            output = input[ix]
            offset += -1
        elif (alphabet.find(input[ix])) > (len(alphabet) - (alphabet.find(key
[((ix + offset) % len(key))]))) - 1):
```

```
            output = alphabet[(alphabet.find(
                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))]))
) % 33]
        else:
            output = alphabet[alphabet.find(
                input[ix]) + (alphabet.find(key[((ix + offset) % len(key))]))
]
        encoded += output
    return encoded


# вывод результатов работы программы
print(f'''
Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{bellaso_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(bellaso_decode(bellaso_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{bellaso_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(bellaso_decode(bellaso_encode(
    input_for_cipher_long(), key), key))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab02_5_bellaso.py
Введите ключ: ключик

Шифр Тритемия:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
мьгдзтъюнзсцунщачэцфатцпспсйапцъаьукэги

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
мържщучроиыкэзжеиэёээоььушаёфщмюхвёэщпмиыкэъхеччкчгеехухрьуьэунйчъэфкчфжшънём
ащпэрсфочэвиыэъхьуэщнюзчммфлйнышррафучлбчрушлуафуочэенлщчъпсаущтёщчквжёцшёбнк
йцуцюнсфэгищыкхъкйнхьюгзнохъятккпюяёфпппакюуцфрзоакмёчяпмфмшдвшъмысшъъвяинщчм
щчхэгиеччщтлёслпульлщэгиеиэёээоььушаёфщмьгвччпщвёккшъжишщцзёёккэзмысшучжыккхч
```

ьоиущплкукыюжеьэвцрьуьэщюйдьйгсисчмъйёкзэъпвчцжцмжщучроечьцъа?иыкэфпйсхкымвит
ёнюьытъюхйчэёээоимхчьоипэнпьййьюмжзэжпгизэучжыкпьюжифщмэоьмшпхаьфувфлтывхщмяш
эпэйарцщянёыыпмйцыжъьгыфщнлкаръээмхркчфжыщюнфкаакьюэдсыпгжеищофлафуонюисчмъйч
ръэюмвчцуггиымщэйёкшпфёдншшъаёрыкэрчнээгищущъфочтэпьпвчфорэйнцжщмиыуъьжезэщэх
аыкэзртъйвфпжщцлрйчхуучжшнтэгикапэыоёйпцъаккпцфхаккпюмшгпчюгвъэкыоахпыщмеиьэъ
жгсомрпйсьушаёфщмфкьцшщэрёфжхъочрчёьюямпцкгдъцщнюикщлъведчъьмиыыкщпйкщчюхвъву
юуйеъыъяьфётличрвуцженцимэйръэююукхирёшюьюмьхпьюмйахщплчущшриёыщыжглсычжжшсы
сфаамйээнзимппйакёчэрчкуэзпйчучъпйеткющизвюэждкщцъаишыщмггичууннйъвуюющшщьчгыц
упнююцёчийьхпшюмдукврпйкпшщмъчмщэнзсйэфэйахьъбгиьуюгиетъюхаыкэзпгсэшжзйнхьюяь
рпофлёлщъьмжььхлёжышуцрёцплявьыэвцлёйщцзцацьэнсеьсшлфьцктлртъйвяёеихщняьръыъя
ьфщмюхв

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов?статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностоо
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.

Блок С: ШИФРЫ БЛОЧНОЙ ЗАМЕНЫ

8. Матричный шифр

Шифр Хилла — полиграммный шифр подстановки, основанный на линейной алгебре и модульной арифметике. Изобретён американским математиком Лестером Хиллом в 1929 году. Это был первый шифр, который позволил на практике (хотя и с трудом) одновременно оперировать более чем с тремя символами. Шифр Хилла не нашёл практического применения в криптографии из-за слабой устойчивости ко взлому и отсутствия описания алгоритмов генерации прямых и обратных матриц большого размера.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import numpy as np
from egcd import egcd

inp = input('Введите матрицу в строку через пробел: ')
inp = inp.split(' ')

# 3 10 20 20 19 17 23 78 17

key = np.matrix([[int(inp[0]), int(inp[1]), int(inp[2])], [int(inp[3]), int(
    inp[4]), int(inp[5])], [int(inp[6]), int(inp[7]), int(inp[8])]])
letter_to_index = dict(zip(alphabet, range(len(alphabet))))
index_to_letter = dict(zip(range(len(alphabet)), alphabet))


def matrix_mod_inv(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = egcd(det, modulus)[1] % modulus
    matrix_modulus_inv = (
        det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    )
    return matrix_modulus_inv


def matrix_encode(message, K):
    encrypted = ""
    message_in_numbers = []
    for letter in message:
        message_in_numbers.append(letter_to_index[letter])

    split_P = [
        message_in_numbers[i: i + int(K.shape[0])]
        for i in range(0, len(message_in_numbers), int(K.shape[0]))
    ]
    for P in split_P:
```

```
        P = np.transpose(np.asarray(P))[:, np.newaxis]
        while P.shape[0] != K.shape[0]:
            P = np.append(P, letter_to_index[" "])[:, np.newaxis]
        numbers = np.dot(K, P) % len(alphabet)
        n = numbers.shape[0]
        for idx in range(n):
            number = int(numbers[idx, 0])
            encrypted += index_to_letter[number]
    return encrypted


def matrix_decode(cipher, Kinv):
    decrypted = ""
    cipher_in_numbers = []
    for letter in cipher:
        cipher_in_numbers.append(letter_to_index[letter])
    split_C = [
        cipher_in_numbers[i: i + int(Kinv.shape[0])]
        for i in range(0, len(cipher_in_numbers), int(Kinv.shape[0]))
    ]
    for C in split_C:
        C = np.transpose(np.asarray(C))[:, np.newaxis]
        numbers = np.dot(Kinv, C) % len(alphabet)
        n = numbers.shape[0]

        for idx in range(n):
            number = int(numbers[idx, 0])
            decrypted += index_to_letter[number]
    return decrypted
# вывод результатов работы программы
print(f'''
Матричный шифр:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{matrix_encode(input_for_cipher_short(), key).replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(matrix_decode(matrix_encode(
    input_for_cipher_short(), key), matrix_mod_inv(key, len(alphabet)))).repl
ace(' ', '')}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{matrix_encode(input_for_cipher_long(), key).replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(matrix_decode(matrix_encode(
    input_for_cipher_long(), key), matrix_mod_inv(key, len(alphabet)))).repla
ce(' ', '')}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_8_matrix.py
Введите матрицу в строку через пробел: 3 10 20 20 19 17 23 78 17

Матричный шифр:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
дёьисжнбнжбеьнцмёаэгщсъттлюцгнхосцгфжгн

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
щвичнкфящёжщрээншусзуйдътюцбёэъяшщктыёжщктийдмпжбзярюмсигфохжртичаужбвфэовккв
кыкзчвзяжтиьудюяшгмювжаепыэсофрдюейхьёзпфрдчвзйсоюбёъштифчыхвлицхилщндкыбоктт
жжгнщъябллфпыдчикъллвусфвпъбёъётстрэуижлиатйчлихчрозюпдмзмлёъзпжюцбёввижгнйдм
фшжтлйбншедщжгнншусзуйдътюцбёдёьбллющрюяшядмдбмйихюяшбээзмлфрдяпйшцпнфшшвкающ
мдуйюьсщигфъарыгхзчэдкхтщтяшцтиоььфчлчнкфящзфэхосжгнёжщюжръчасожцповусштдбнив
щусзйдзшцпвфшьйотрцачиаъшцъйёгфрдэъпчияъжпшеьёттскиенаеэёгсюъвуефъацжщофабюяз
ърцпяшлнкчтшюистщоъшхгчкгёядьчтйёьчеажщъёяъьявшзмлфрдяпйэдкхтщщплгйяоььъжхлжщ
цэоеяшъбьоиыфэцъаптъхйжгнзыгвядгобомиььхеёжкрмзпчичнкыфмшэомюцмээусзлеэзнкфме
дщкфрдимежгнщцгмфвтэяхосймфыгщмнцьвпёдцщэобзчожтвнокдызпмвщвгщзыюжрътюцбёпчръ
щръмяюбёмллаьпяыогнобрчгюяшажиьгдмжзнкёчичыдгъяьиммсшмюцфммрэнохюёжюпплкяыщбя
ьъиохжиъожфцшщъщьачижччачижгншвквхшёсошязцезрэлцъсшдыоемъсимзецарцётьнцфжуэфг
ъмммвщебжюммвфудпулкктиэхосйбмтэячтфтщогземюцлиьвъжщцкбвжяьбнфжччхлтмъбёючжъя
ётгьеобэобеежржгнщпгхвдкнёыъчдбнмюцъммэуилэчфпыщчишюззмлбасзнкэучцпозбнжяымещ
кщтжвпуипоэяйщерджбйозюьщштщрэжюусзчйдчшрозжншрэнъенэъягнх

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностоль
коразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговоспр
иятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

9. Шифр Плейфера

Шифр Плейфера или квадрат Плейфера — ручная симметричная техника шифрования, в которой впервые использована замена биграмм. Изобретена в 1854 году английским физиком Чарльзом Уитстоном, но названа именем лорда Лайона Плейфера, который внёс большой вклад в продвижение использования данной системы шифрования в государственной службе. Шифр предусматривает шифрование пар символов (биграмм) вместо одиночных символов, как в шифре подстановки и в более сложных системах шифрования Виженера. Таким образом, шифр Плейфера более устойчив к взлому по сравнению с шифром простой замены, так как усложняется его частотный анализ. Он может быть проведён, но не для символов, а для биграмм. Так как возможных биграмм больше, чем символов, анализ значительно более трудоёмок и требует большего объёма зашифрованного текста.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

alphabet = alphabet.replace(' ', '') + 'abc'

key = str(input('Введите ключ: '))


def playfair_encode(clearText, key):
    text = clearText
    new_alphabet = []
    for i in range(len(key)):
        new_alphabet.append(key[i])
    for i in range(len(alphabet)):
        bool_buff = False
        for j in range(len(key)):
            if alphabet[i] == key[j]:
                bool_buff = True
                break
        if bool_buff == False:
            new_alphabet.append(alphabet[i])
    mtx_abt_j = []
    counter = 0
    for j in range(6):
        mtx_abt_i = []
        for i in range(6):
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    if len(text) % 2 == 1:
```

```
        text = text + "я"
    enc_text = ""
    for t in range(0, len(text), 2):
        flag = True
        for j_1 in range(6):
            if flag == False:
                break
            for i_1 in range(6):
                if flag == False:
                    break
                if mtx_abt_j[j_1][i_1] == text[t]:
                    for j_2 in range(6):
                        if flag == False:
                            break
                        for i_2 in range(6):
                            if mtx_abt_j[j_2][i_2] == text[t+1]:
                                if j_1 != j_2 and i_1 != i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][i_2] + \
                                        mtx_abt_j[j_2][i_1]
                                elif j_1 == j_2 and i_1 != i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][(i_1+1) % 6] + \
                                        mtx_abt_j[j_2][(i_2+1) % 6]
                                elif j_1 != j_2 and i_1 == i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[(j_1+1) % 5][i_1] + \
                                        mtx_abt_j[(j_2+1) % 5][i_2]
                                elif j_1 == j_2 and i_1 == i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][i_1] + \
                                        mtx_abt_j[j_1][i_1]
                                flag = False
                                break
    return enc_text


def playfair_decode(clearText, key):
    text = clearText
    new_alphabet = []
    for i in range(len(key)):
        new_alphabet.append(key[i])
    for i in range(len(alphabet)):
        bool_buff = False
        for j in range(len(key)):
            if alphabet[i] == key[j]:
                bool_buff = True
                break
        if bool_buff == False:
            new_alphabet.append(alphabet[i])
```

```
    mtx_abt_j = []
    counter = 0
    for j in range(6):
        mtx_abt_i = []
        for i in range(6):
            mtx_abt_i.append(new_alphabet[counter])
            counter = counter + 1
        mtx_abt_j.append(mtx_abt_i)
    if len(text) % 2 == 1:
        text = text + "я"
    enc_text = ""
    for t in range(0, len(text), 2):
        flag = True
        for j_1 in range(6):
            if flag == False:
                break
            for i_1 in range(6):
                if flag == False:
                    break
                if mtx_abt_j[j_1][i_1] == text[t]:
                    for j_2 in range(6):
                        if flag == False:
                            break
                        for i_2 in range(6):
                            if mtx_abt_j[j_2][i_2] == text[t+1]:
                                if j_1 != j_2 and i_1 != i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][i_2] + \
                                        mtx_abt_j[j_2][i_1]
                                elif j_1 == j_2 and i_1 != i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][(i_1-1) % 6] + \
                                        mtx_abt_j[j_2][(i_2-1) % 6]
                                elif j_1 != j_2 and i_1 == i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[(j_1-1) % 5][i_1] + \
                                        mtx_abt_j[(j_2-1) % 5][i_2]
                                elif j_1 == j_2 and i_1 == i_2:
                                    enc_text = enc_text + \
                                        mtx_abt_j[j_1][i_1] + \
                                        mtx_abt_j[j_1][i_1]
                                flag = False
                                break
    return enc_text
# вывод результатов работы программы
print(f'''
Шифр Плейфера:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{playfair_encode(input_for_cipher_short(), key)}
```

```
Расшифрованный текст:
{output_from_decrypted(playfair_decode(playfair_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{playfair_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(playfair_decode(playfair_encode(
    input_for_cipher_long(), key), key))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab03_9_playfair.py
Введите ключ: ключик

Шифр Плейфера:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
зцднэйрурскююдакпуюкаэунмбоукгкпгааёсичы

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
аруртюндстодпакмдояогготдтаркпдрилартаптодупкйтёбкнуылчмудчормрупрщдёдббёурпб
щтаашчмбиычвоупкгиораворадюмувуунщдюкёддбмюёещлюкбиьйавпкяхлщкмкфтзвфктнньфро
жбклвфчмсиюаодафтщактувуаеафаьгбдуапкбаеерщлюкусжфбвёвъвдюпаякёутакмрпвмбдпкр
афасикётёпёёулдвнунассикёдояогготдтаркпзцактёнуатгбёучтрпббёргбпатакмкюдмгблю
чидктауъаёсукмоуилудчоумдояогготдтаркпзвупочпкылпртюндузптктпдрилтудочтщдаёрпа
ёёэгбдужрсиупоашгкгаюючкгдугравгшупсьпаеашгщдюкеггущдпчратсаеунзгбккикмаоилёу
жрудпчюмкпорпууважбасбхувбисадёюмрутпврдёиивтседткигосатдувикёетакмкюдмгбтчздп
квёруупафюккгтуарпчраунюмндннрарёовтубаттилаюпрютгёудстафмггbудббёутулттюйbуп
шгдщдоапяоггчтрспабкдёииюквамрилскдурспабкраребккиюдбаупехднудчоодрсдтвуёуёет
уткюкеггущдтчздпкрадтнуёутупкылпсвёёаовмвбкbгйткпгбргпатаёbжтсптуовйурдтёсичо
икодпарспабкэёёавёиклкунючгьрмруодккёаарпротупднгуупсиафемёауёакпупсbащктзакд
лпзюддмастровгабиюдаётубгдщяпупдтптпаёвоашгкстчздпкратрспвакбтджрутикодьспткб
емкдгбзёаёьюдннуупёигкгурднуёуасартртюасчасичосакбтчудпяжрсидщдояпюкумяёудчоп
двнаекмсапрспроочвёрумкиоуёавтедусикёпапкяхкмтуеруъзёвфнувёдояоггрнёеафгввнрс
пабкрасичы

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
```

```
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименносто
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

D: ШИФРЫ ПЕРЕСТАНОВКИ

10.Шифр вертикальной перестановки

Широкое распространение получила разновидность маршрутной перестановки — вертикальная перестановка. В этом шифре также используется прямоугольная таблица, в которую сообщение записывается по строкам слева направо. Выписывается шифрограмма по вертикалям, при этом столбцы выбираются в порядке, определяемом ключом.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import math

key = str(input('Введите ключ: '))

def transposition_encode(msg, key):
    cipher = ""

    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    col = len(key)

    row = int(math.ceil(msg_len / col))

    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    matrix = [msg_lst[i: i + col] for i in range(0, len(msg_lst), col)]

    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx] for row in matrix])
        k_indx += 1

    return cipher


def transposition_decode(cipher, key):
    msg = ""

    k_indx = 0

    msg_indx = 0
```

```
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    col = len(key)

    row = int(math.ceil(msg_len / col))

    key_lst = sorted(list(key))

    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]

    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

        for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1

    null_count = msg.count('_')

    if null_count > 0:
        return msg[: -null_count]

    msg = ''.join(sum(dec_cipher, []))

    return msg.replace('_', '')

# вывод результатов работы программы
print(f'''
Шифр вертикальной перестановки:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{transposition_encode(input_for_cipher_short(), key)}

Расшифрованный текст:
{output_from_decrypted(transposition_decode(transposition_encode(
    input_for_cipher_short(), key), key))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{transposition_encode(input_for_cipher_long(), key)}

Расшифрованный текст:
{output_from_decrypted(transposition_decode(transposition_encode(
    input_for_cipher_long(), key), key))}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab04_10_transposition.py
Введите ключ: ключ

Шифр вертикальной перестановки:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
вяпиовжчвтрзрвтыдеечмтлииетоа_епиылнулкк

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
врртаяилчотчаьттомндядачооннлгнляоифанпиикеекквбехтацонидоокоинтаяилеевилвоив
юокичкаяилтоомовсикквзтсвчвяпдтдтоейиынтиуряроисадиамчоивматлсснморачоарйтнин
саыиоалзупеуиабттинслесоиноомзялсомснокпеачнбпкэуеткаеофырисвиситозссосбмтто
двыетатнвриклтзиьтттеорспкеекоивжеаянвпетоисьтчмокданлкезпаохщлреввтеиааинлхо
цнукйвокроаодирбебонзлкнжбечтчмокнасьадлачданкстчмоолпесттсаааповякассяеививдв
чтоезпетегзомрмсиидлавзоитлееваекпйсдеопячтссбмитчрлвчеъеамотисилмолрыдеовдпт
смсарлкиеятктсмооккриижднпевттосаяилпеисаснаммочвооиясаептснебдгокттбтнлнуннтч
абрлчпеанссоттсомнйстиьодйкотририаиилбшнмохлцчаттдытеуихаичдогвчмозонссормооо
оьндлиуттенссоэккиноктиоытчыаютбоьяислрелнкплотлплмтзигчяеандилтоеовзнзтткрео
яьтиоттчрлиектбвлвомсрраививвнткзалсабыоавчиьбззиюзакпосчнntемивталмвсмьыувво
апипеенлнксноптчгиьчтийсзнпузиндчбштуцзсзозбв_тмтиыуввэотоеикптлпоияткаветмзх
деьириыбаттмсебелвлёзвыопаотонегкыувводнпзтиикануттыуввсьррлчатпзеттчлееттсл
есснеичзслобьдапюиуитрниисопкчвоиностьвиткелсртиьяпеибчеооеитекпенодтмоесьарем
вонрртчтоыаклттаотетдооырбиярдыаьиттчморлзчялижэемеегсятоссталыкеиопаноутоьс
нааыукеоок

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименносто
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибирживидятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

11.Решетка Кардано

Решётка Кардано — исторически первая известная шифровальная решётка, трафарет, применявшийся для шифрования и дешифрования, выполненный в форме прямоугольной (чаще всего — квадратной) таблицы-карточки, часть ячеек которых вырезана, и через которые наносился шифротекст. Пустые поля текста заполнялись другим текстом для маскировки сообщений под обычные послания — таким образом, применение решётки является одной из форм стеганографии.

Код программы:

```
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

class Cardan(object):
    def __init__(self, size, spaces):
        self.size = int(size)
        str1 = ''
        for i in range(len(spaces)):
            str1 = str1 + str(spaces[i][0]) + str(spaces[i][1])
        self.spaces = str1
        matrix_spaces = []
        i = 0
        cont = 0
        while i < self.size*self.size//4:
            t = int(self.spaces[cont]), int(self.spaces[cont + 1])
            cont = cont + 2
            i = i+1
            matrix_spaces.append(t)
        self.spaces = matrix_spaces

    def code(self, message):
        offset = 0
        cipher_text = ""
        matrix = []
        for i in range(self.size*2-1):
            matrix.append([])
            for j in range(self.size):
                matrix[i].append(None)
        whitesneeded = self.size*self.size - \
            len(message) % (self.size*self.size)
        if (len(message) % (self.size*self.size) != 0):
            for h in range(whitesneeded):
                message = message + ' '
        while offset < len(message):
            self.spaces.sort()
            for i in range(int(self.size*self.size//4)):
                xy = self.spaces[i]
```

```
                x = xy[0]
                y = xy[1]
                matrix[x][y] = message[offset]
                offset = offset + 1
            if (offset % (self.size*self.size)) == 0:
                for i in range(self.size):
                    for j in range(self.size):
                        try:
                            cipher_text = cipher_text + matrix[i][j]
                        except:
                            pass
            for i in range(self.size*self.size//4):
                x = (self.size-1)-self.spaces[i][1]
                y = self.spaces[i][0]
                self.spaces[i] = x, y
        return cipher_text


    def decode(self, message, size):
        uncipher_text = ""
        offset = 0
        matrix = []
        for i in range(self.size*2-1):
            matrix.append([])
            for j in range(self.size):
                matrix[i].append(None)
        whitesneeded = self.size*self.size - \
            len(message) % (self.size*self.size)
        if (len(message) % (self.size*self.size) != 0):
            for h in range(whitesneeded):
                message = message + ' '
        offsetmsg = len(message) - 1
        while offset < len(message):
            if (offset % (self.size*self.size)) == 0:
                for i in reversed(list(range(self.size))):
                    for j in reversed(list(range(self.size))):
                        matrix[i][j] = message[offsetmsg]
                        offsetmsg = offsetmsg - 1
            for i in reversed(list(range(self.size*self.size//4))):
                x = self.spaces[i][1]
                y = (self.size-1)-self.spaces[i][0]
                self.spaces[i] = x, y
            self.spaces.sort(reverse=True)
            for i in range(self.size*self.size//4):
                xy = self.spaces[i]
                x = xy[0]
                y = xy[1]
                uncipher_text = matrix[x][y] + uncipher_text
                offset = offset + 1
```

```
        return uncipher_text


gaps = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1), (7, 2
), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]
r = Cardan(8, gaps)

texto = input_for_cipher_short()

n = len(texto)
encoded = r.code(texto)
decoded = r.decode(encoded, n)

gaps2 = [(7, 7), (6, 0), (5, 0), (4, 0), (7, 1), (1, 1), (1, 2), (4, 1), (7,
2), (2, 1), (2, 5), (2, 3), (7, 3), (3, 1), (3, 2), (3, 4)]
r2 = Cardan(8, gaps)

texto_long = input_for_cipher_long()

n = len(texto_long)
encoded_long = r2.code(texto_long)
decoded_long = r2.decode(encoded_long, n)

print(f'''
Шифр вертикальной перестановки:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encoded.replace(' ', '')}

Расшифрованный текст:
{output_from_decrypted(decoded)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encoded_long}

Расшифрованный текст:
{output_from_decrypted(decoded_long)}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab04_11_cardan.py

Шифр вертикальной перестановки:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
векаоврtетмячзплткпривиынелждутчивыелои
```

```
Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
чалекэтанвоькийоттгперкдсимыетосрсстзатятчоучпнсаимвоомлттьиовтнолимваряаопга
зионтщиамхвиалиьлвинойнитедплряндекоартотичлдхоектдиерейтчадянкобыквецбаоевтл
ьишбтаихокоомнинтыелкхнпублстиеформацмдожнзагцоедибезонвеуехглоилвитовтриочкт
оёбчыккнчхноодоминабзнпоалвакьзоелатючаивоыксдянаучуосктьиммоадиевнндирлоован
оидитловспоровтчлькякинстатоиучтсчтпиктуекрикссмаеритмнвопсонловэлотоатыоскся
двестосскатислпозяыввсяраечтетьзпаддесвткялтниюччаетливетотсебыясоютьптзйвами
ирдеелриуегчиснидлытломгаичзмлиизокупотптрчнозеблптнеиокоизасменнлотлявмоизир
иерчечидсатввнаоссслиамволоватодизптниысяприрчаеиспрнотстбчеялквккатоопосмчий
идтеаитяртелььтниайтостебнонъемеалистоитлблеиздевтчокексучвтсапуевреитмлтичив
ераипроетобямпремсоосистралнмлвсотовлоьвоваимксмвооербаотдзнмыранызчносделтпе
меттаасксстотчккчкиотдкнатаьаакпрзэкточоипкуосибнелютобнелыятззиоствымрьекзаты
ссожтяочтурыиесавфивиитьирдсятмотмспраиммвыибедливачеажнтсолтвеныносвосгмопрч
овэлбеиеометланятсопаослемкдпмизниептогоныйтприроПутсяеткиаезятсчпкскскьттбезо
пзтендчгитатиньиласслииевбеназозктпробаеолнлеотвбуьдтысетшяччуитнзснкатчвунук
ож кнонацб

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностоо
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

Е: ШИФРЫ ГАММИРОВАНИЯ

13.Одноразовый блокнот К.Шеннона

Популярность поточных шифров можно связывать с работой Клода Шеннона, посвященной анализу одноразовых гамма-блокнотов. Название «одноразовый блокнот» стало общепринятым в годы Второй мировой войны, когда для шифрования широко использовались бумажные блокноты.

Одноразовый блокнот использует длинную шифрующую последовательность, которая состоит из случайно выбираемых бит или наборов бит (символов). Шифрующая последовательность побитно или посимвольно накладывается на открытый текст, имеет ту же самую длину, что и открытое сообщение, и может использоваться только один раз (о чем свидетельствует само название шифрсистемы); ясно, что при таком способе шифрования требуется огромное количество шифрующей гаммы.

Открытый текст сообщения ш записывают как последовательность бит или символов m = momi...mn_i, а двоичную или символьную шифрующую последовательность к той же самой длины - как k = koki...k,,_|.

Шифртекст c = c0cl...cn.i определяется соотношением Cj = mi Шк, при 0

Код программы:

```
import random
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted

alphabet = alphabet.replace(' ', '')
alphabet_lower = {}
i = 0
while i < (len(alphabet)):
    alphabet_lower.update({alphabet[i]: i})
    i += 1
def get_key(d, value):
    for k, v in d.items():
        if v == value:
            return k


def shenon_encode(msg):
    msg_list = list(msg)
    msg_list_len = len(msg_list)
    msg_code_bin_list = list()
    for i in range(len(msg_list)):
        msg_code_bin_list.append(alphabet_lower.get(msg_list[i]))

    key_list = list()
```

```
    for i in range(msg_list_len):
        key_list.append(random.randint(0, 32))

    cipher_list = list()
    for i in range(msg_list_len):
        m = int(msg_code_bin_list[i])
        k = int(key_list[i])
        cipher_list.append(int(bin(m ^ k), base=2))
    return cipher_list, key_list


def shenon_decode(msg, key_list):
    decipher_list = list()
    msg_list_len = len(msg)
    for i in range(msg_list_len):
        c = int(msg[i])
        k = int(key_list[i])
        decipher_list.append(int(bin(c ^ k), base=2))
    deciphered_str = ""
    for i in range(len(decipher_list)):
        deciphered_str += get_key(alphabet_lower, decipher_list[i])
    return deciphered_str


short_encoded = shenon_encode(input_for_cipher_short())
short_decoded = shenon_decode(short_encoded[0], short_encoded[1])

long_encoded = shenon_encode(input_for_cipher_long())
long_decoded = shenon_decode(long_encoded[0], long_encoded[1])

print(f'''
Одноразовый блокнот:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{short_encoded[0]}
Ключ:
{short_encoded[1]}

Расшифрованный текст:
{output_from_decrypted(short_decoded)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{long_encoded[0]}
Ключ:
{long_encoded[1]}

Расшифрованный текст:
{output_from_decrypted(long_decoded)}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab05_13_shenon.py

Одноразовый блокнот:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
[20, 24, 3, 1, 54, 21, 5, 27, 10, 6, 19, 22, 12, 23, 30, 22, 12, 20, 31, 13,
30, 6, 17, 14, 21, 0, 0, 4, 20, 10, 7, 10, 0, 3, 15, 9, 26, 31, 26]
Ключ:
[22, 9, 6, 12, 22, 29, 21, 8, 26, 23, 26, 26, 5, 21, 2, 31, 3, 7, 19, 4, 28,
26, 31, 11, 18, 4, 20, 23, 12, 15, 11, 5, 2, 6, 4, 9, 9, 7, 17]

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
[12, 47, 30, 29, 1, 28, 12, 13, 17, 28, 14, 8, 8, 13, 23, 25, 7, 14, 25, 7,
49, 56, 27, 11, 13, 16, 10, 24, 9, 14, 3, 26, 22, 6, 9, 6, 7, 0, 6, 5, 7, 31,
7, 12, 12, 11, 1, 15, 32, 25, 24, 4, 31, 17, 7, 9, 2, 8, 23, 9, 17, 17, 48,
4, 3, 19, 1, 10, 1, 4, 27, 9, 25, 23, 15, 17, 29, 12, 13, 3, 47, 21, 22, 12,
19, 19, 58, 10, 8, 12, 11, 14, 31, 31, 31, 9, 6, 18, 1, 49, 3, 10, 27, 3, 19,
21, 26, 16, 13, 2, 5, 21, 15, 26, 5, 5, 5, 32, 29, 28, 9, 14, 25, 3, 22, 15,
6, 14, 34, 14, 7, 15, 18, 0, 29, 1, 27, 14, 12, 46, 16, 27, 2, 4, 27, 16, 16,
10, 12, 0, 21, 1, 28, 24, 17, 1, 8, 27, 19, 13, 11, 16, 20, 9, 1, 34, 23, 19,
25, 7, 20, 11, 1, 14, 23, 9, 10, 21, 4, 10, 29, 6, 3, 6, 6, 9, 23, 0, 4, 0,
14, 14, 9, 12, 15, 1, 20, 14, 27, 5, 20, 0, 13, 13, 25, 27, 1, 7, 7, 2, 15,
6, 7, 27, 10, 23, 2, 9, 31, 21, 22, 25, 22, 23, 36, 23, 31, 10, 18, 8, 19,
28, 21, 8, 4, 23, 1, 22, 29, 31, 28, 27, 31, 22, 21, 3, 23, 30, 7, 11, 0, 18,
31, 20, 4, 20, 25, 27, 19, 8, 45, 26, 9, 14, 7, 16, 20, 15, 23, 26, 11, 16,
29, 23, 7, 24, 8, 4, 4, 9, 9, 14, 22, 19, 15, 23, 3, 2, 15, 9, 19, 1, 9, 11,
9, 17, 14, 27, 12, 25, 7, 21, 2, 8, 31, 13, 23, 44, 13, 5, 9, 29, 23, 20, 11,
25, 2, 5, 2, 4, 26, 23, 20, 27, 9, 27, 30, 24, 14, 7, 17, 23, 4, 0, 13, 17,
33, 22, 10, 20, 9, 13, 24, 16, 27, 1, 3, 11, 21, 9, 17, 20, 17, 20, 19, 26,
21, 24, 1, 17, 0, 9, 22, 9, 19, 4, 44, 22, 25, 28, 26, 28, 5, 29, 12, 3, 12,
8, 22, 18, 5, 7, 14, 8, 27, 16, 12, 15, 29, 30, 6, 20, 17, 29, 15, 14, 18,
24, 7, 22, 28, 38, 15, 28, 6, 16, 4, 6, 21, 23, 9, 22, 9, 1, 18, 26, 52, 22,
19, 21, 25, 37, 15, 28, 31, 8, 30, 36, 19, 17, 14, 12, 26, 5, 4, 30, 6, 0, 5,
25, 0, 14, 25, 27, 1, 36, 14, 15, 6, 11, 31, 13, 17, 1, 31, 27, 28, 7, 12,
22, 8, 28, 17, 22, 16, 25, 25, 28, 21, 29, 13, 0, 20, 17, 13, 23, 23, 37, 15,
8, 58, 4, 8, 9, 27, 7, 5, 19, 21, 13, 21, 25, 24, 19, 4, 2, 26, 25, 12, 11,
24, 26, 8, 19, 11, 11, 31, 35, 1, 9, 26, 22, 2, 25, 26, 60, 24, 2, 49, 15,
22, 27, 14, 3, 29, 12, 7, 13, 7, 31, 2, 12, 22, 13, 14, 17, 5, 6, 31, 21, 21,
14, 14, 25, 4, 7, 30, 25, 27, 10, 8, 16, 7, 14, 6, 29, 29, 17, 2, 31, 15, 12,
14, 19, 17, 31, 21, 25, 16, 10, 26, 19, 9, 31, 22, 15, 32, 18, 1, 27, 3, 0,
17, 5, 47, 10, 11, 16, 4, 29, 4, 4, 17, 21, 10, 14, 10, 10, 13, 50, 15, 15,
2, 19, 15, 10, 18, 7, 29, 48, 13, 15, 13, 42, 7, 1, 13, 7, 0, 27, 4, 0, 16,
13, 17, 4, 34, 1, 12, 7, 6, 8, 19, 13, 10, 9, 22, 25, 10, 23, 12, 23, 30, 29,
40, 10, 16, 2, 16, 2, 7, 13, 31, 27, 15, 16, 11, 0, 19, 4, 10, 31, 26, 16,
```

```
24, 56, 2, 18, 8, 0, 30, 24, 5, 11, 0, 6, 31, 19, 10, 31, 8, 23, 5, 24, 26,
6, 30, 11, 13, 25, 16, 17, 13, 6, 30, 21, 15, 7, 1, 9, 2, 15, 12, 14, 0, 10,
6, 31, 5, 29, 26, 5, 30, 12, 28, 4, 18, 8, 1, 13, 24, 24, 7, 8, 21, 27, 25,
17, 18, 2, 27, 16, 3, 4, 12, 22, 45, 15, 0, 10, 14, 29, 34, 31, 12, 22, 31,
21, 26, 2, 16, 16, 11, 15, 0, 6, 23, 26, 25, 2, 2, 28, 16, 3, 24, 2, 25, 17,
26, 10, 26, 16, 21, 25, 27, 4, 18, 21, 23, 6, 0, 18, 30, 6, 24, 17, 1, 16,
25, 22, 20, 15, 30, 37, 29, 4, 21, 54, 6, 18, 3, 8, 6, 12, 23, 27, 15, 27, 4,
51, 29, 30, 7, 14, 31, 5, 0, 18, 18, 29, 31, 5, 27, 19, 21, 30, 4, 4, 21, 31,
31, 46, 3, 16, 18, 4, 12, 25, 5, 22, 28, 7, 11, 9, 29, 23, 29, 22, 11, 12,
10, 7, 24, 29, 35, 7, 13, 10, 12, 12, 17, 17, 1, 14, 17, 22, 19, 21, 20, 16,
0, 20, 17, 51, 13, 30, 10, 24, 2, 25, 20, 21, 22, 17, 13, 23, 13, 25, 29, 41,
25, 20, 26, 23, 25, 12, 20, 1, 26, 34, 11, 16, 20, 29, 20, 17, 44, 5, 21, 13,
27, 7, 17, 50, 21, 0, 13, 31, 53, 31, 27, 21, 19, 26, 24, 17, 41, 5, 30, 21,
8, 14, 16, 45, 3, 22, 37, 17, 21, 20, 25, 12, 45, 26, 14, 14, 1, 50, 18, 21,
22, 5, 18, 24, 17, 24, 13, 28, 21, 10, 16, 6, 52, 1, 30, 35, 17, 23, 2, 25,
1, 20, 19, 9, 20, 7, 18, 26, 3, 9, 11, 31, 11, 22, 26, 7, 11, 15, 16, 31, 20,
8, 11, 20, 11, 5, 0, 27, 24, 1, 9, 30, 22, 23, 14, 7, 41, 28, 31, 16, 30, 21,
10, 28, 19, 10, 14, 25, 10, 2, 19, 21, 27, 0, 11, 26, 26, 27, 30, 12, 21, 4,
10, 0, 10, 25, 13, 11, 20, 13, 31, 19, 27, 15, 29, 22, 8, 21, 31, 21, 11, 9,
0, 15, 10, 7, 4, 17, 14, 16, 24, 9, 22, 3, 55, 17, 3, 6, 46, 23, 21, 14, 18,
18, 1, 4, 18, 25, 25, 0, 16, 7, 8, 19, 51, 25, 30]
Ключ:
[14, 32, 13, 13, 16, 21, 1, 8, 0, 14, 29, 8, 27, 16, 30, 23, 7, 29, 5, 21,
17, 32, 15, 25, 4, 29, 8, 23, 5, 1, 1, 9, 14, 13, 23, 21, 8, 4, 9, 23, 20,
31, 20, 3, 20, 5, 14, 2, 32, 21, 29, 10, 2, 26, 14, 3, 17, 13, 28, 27, 2, 25,
32, 23, 12, 3, 18, 3, 12, 4, 23, 20, 23, 24, 31, 30, 25, 26, 2, 7, 15, 15,
31, 6, 23, 31, 26, 1, 8, 29, 24, 1, 7, 26, 20, 26, 9, 16, 1, 32, 12, 8, 25,
10, 29, 6, 31, 1, 3, 7, 22, 28, 3, 19, 8, 5, 6, 32, 21, 21, 7, 14, 15, 10,
26, 6, 2, 2, 2, 0, 2, 14, 29, 12, 0, 24, 18, 24, 5, 32, 5, 20, 19, 9, 27, 7,
25, 5, 2, 14, 9, 23, 12, 12, 16, 13, 1, 16, 19, 26, 2, 26, 7, 17, 10, 32, 4,
19, 18, 8, 25, 24, 4, 5, 5, 26, 15, 4, 1, 14, 22, 9, 2, 26, 4, 9, 18, 19, 5,
15, 2, 11, 12, 8, 13, 21, 2, 7, 23, 12, 7, 17, 11, 27, 25, 26, 9, 7, 16, 7,
13, 15, 8, 26, 22, 15, 12, 6, 16, 17, 31, 23, 6, 24, 32, 31, 31, 9, 29, 4,
28, 30, 26, 3, 23, 15, 10, 24, 18, 18, 19, 28, 17, 25, 28, 2, 18, 22, 9, 14,
3, 29, 12, 12, 15, 26, 25, 8, 15, 26, 13, 2, 29, 28, 14, 29, 22, 0, 27, 21,
9, 1, 24, 28, 8, 21, 13, 10, 0, 6, 11, 14, 24, 28, 6, 5, 19, 13, 3, 20, 27,
14, 11, 11, 26, 12, 1, 31, 5, 23, 14, 25, 11, 12, 29, 13, 28, 32, 18, 29, 9,
20, 24, 16, 5, 13, 9, 5, 19, 23, 19, 25, 0, 8, 17, 16, 13, 29, 5, 21, 2, 25,
4, 19, 17, 3, 1, 14, 30, 6, 0, 0, 26, 31, 23, 14, 1, 21, 6, 6, 3, 31, 30, 24,
14, 17, 26, 8, 16, 24, 13, 12, 7, 7, 28, 22, 32, 25, 27, 15, 2, 23, 23, 14,
12, 16, 5, 26, 5, 27, 14, 7, 30, 7, 16, 16, 4, 19, 31, 30, 3, 7, 25, 13, 28,
22, 1, 23, 20, 10, 14, 6, 23, 28, 4, 27, 8, 25, 13, 23, 12, 5, 11, 19, 23,
27, 20, 4, 0, 26, 9, 5, 28, 1, 27, 13, 12, 4, 0, 24, 2, 5, 30, 7, 1, 12, 21,
9, 23, 21, 15, 12, 11, 10, 4, 32, 0, 10, 12, 9, 26, 1, 24, 25, 22, 21, 0, 20,
20, 29, 6, 19, 25, 6, 3, 28, 11, 16, 28, 21, 1, 15, 0, 1, 2, 4, 6, 32, 14, 4,
26, 23, 21, 25, 10, 2, 1, 31, 26, 14, 21, 20, 17, 27, 20, 17, 8, 22, 19, 3,
24, 23, 1, 26, 15, 26, 11, 32, 8, 4, 19, 14, 2, 11, 9, 28, 21, 11, 32, 10,
14, 18, 0, 3, 18, 8, 14, 3, 14, 19, 11, 8, 20, 13, 28, 24, 8, 4, 16, 25, 21,
6, 30, 10, 23, 8, 21, 22, 23, 3, 16, 21, 21, 29, 4, 18, 15, 29, 13, 29, 1, 9,
7, 27, 28, 26, 27, 23, 31, 8, 21, 27, 24, 31, 4, 28, 32, 23, 18, 8, 27, 11,
```

```
19, 14, 32, 26, 2, 1, 4, 23, 23, 1, 0, 7, 1, 1, 0, 14, 8, 18, 28, 10, 14, 14,
1, 5, 0, 20, 20, 32, 28, 6, 3, 10, 20, 14, 31, 31, 9, 8, 4, 19, 13, 30, 13,
22, 2, 25, 5, 21, 22, 25, 28, 12, 15, 5, 22, 20, 3, 30, 0, 30, 31, 24, 32,
25, 8, 9, 4, 26, 2, 30, 15, 10, 0, 17, 14, 12, 28, 6, 30, 29, 31, 28, 17, 32,
11, 16, 8, 5, 13, 23, 4, 16, 5, 11, 12, 22, 1, 13, 27, 23, 21, 9, 19, 11, 27,
26, 3, 22, 30, 17, 31, 21, 17, 28, 3, 14, 5, 11, 7, 29, 31, 7, 18, 3, 11, 29,
10, 17, 21, 7, 23, 1, 25, 10, 28, 7, 19, 30, 23, 20, 26, 3, 26, 10, 25, 25,
31, 30, 10, 16, 11, 0, 9, 26, 13, 10, 13, 24, 2, 18, 32, 31, 30, 20, 16, 20,
21, 6, 30, 12, 6, 31, 17, 9, 5, 9, 8, 2, 12, 14, 3, 1, 23, 15, 10, 9, 17, 24,
2, 25, 6, 25, 8, 25, 2, 4, 24, 7, 5, 30, 2, 14, 24, 26, 1, 24, 1, 31, 31, 6,
16, 32, 17, 27, 20, 22, 21, 26, 19, 27, 21, 12, 28, 16, 15, 16, 26, 32, 18,
14, 19, 28, 12, 10, 5, 31, 23, 15, 12, 10, 8, 11, 30, 17, 0, 10, 21, 20, 16,
32, 6, 27, 29, 23, 3, 8, 25, 19, 9, 14, 26, 4, 1, 30, 28, 31, 26, 11, 3, 5,
17, 25, 3, 20, 31, 26, 29, 12, 19, 20, 5, 2, 24, 20, 15, 24, 6, 3, 0, 22, 24,
32, 16, 12, 25, 23, 11, 20, 27, 7, 5, 12, 5, 23, 30, 5, 15, 9, 1, 0, 8, 30,
20, 14, 27, 13, 21, 32, 25, 0, 5, 18, 21, 20, 32, 5, 24, 4, 19, 23, 2, 32,
13, 9, 30, 31, 21, 15, 20, 7, 31, 31, 28, 31, 32, 0, 28, 21, 15, 0, 12, 32,
29, 26, 32, 28, 16, 26, 10, 3, 32, 17, 14, 22, 4, 32, 1, 23, 19, 11, 28, 23,
18, 23, 15, 19, 7, 26, 1, 15, 20, 18, 23, 3, 2, 15, 9, 11, 14, 23, 31, 9, 6,
14, 1, 31, 17, 20, 3, 15, 24, 14, 19, 20, 11, 28, 13, 13, 24, 1, 24, 26, 23,
15, 19, 30, 19, 19, 26, 31, 19, 31, 11, 3, 32, 18, 16, 19, 17, 5, 27, 19, 3,
30, 28, 18, 10, 10, 3, 6, 21, 9, 0, 9, 21, 21, 27, 13, 1, 0, 15, 19, 25, 1,
6, 5, 27, 12, 16, 31, 6, 22, 20, 24, 26, 6, 29, 1, 5, 29, 7, 1, 10, 16, 1,
31, 14, 24, 24, 26, 10, 17, 23, 9, 23, 14, 32, 23, 30, 1, 16, 19, 4, 12, 2,
8, 22, 1, 21, 11, 7, 17, 32, 1, 21]

Расшифрованный текст:
вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименносто
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржividятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговоспр
иятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```

14.Гаммирование ГОСТ 28147-89

При работе ГОСТ 28147-89 в режиме гаммирования описанным выше образом формируется криптографическая гамма, которая затем побитно складывается по модулю 2 с исходным открытым текстом для получения шифротекста. Шифрование в режиме гаммирования лишено недостатков, присущих режиму простой замены. Так, даже идентичные блоки исходного текста дают разный шифротекст, а для текстов с длиной, не кратной 64 бит, "лишние" биты гаммы отбрасываются. Кроме того, гамма может быть выработана заранее, что соответствует работе шифра в поточном режиме.

Код программы:

```
# -*- coding:utf-8 -*-
import sys
import numpy.random
import itertools
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import binascii


class GostCrypt(object):
    def __init__(self, key, sbox):
        self._key = None
        self._subkeys = None
        self.key = key
        self.sbox = sbox

    @staticmethod
    def _bit_length(value):
        return len(bin(value)[2:])

    @property
    def key(self):
        return self._key

    @key.setter
    def key(self, key):
        self._key = key
        self._subkeys = [(key >> (32 * i)) & 0xFFFFFFFF for i in range(8)]


    def _f(self, part, key):
        temp = part ^ key
        output = 0
        for i in range(8):
            output |= ((self.sbox[i][(temp >> (4 * i)) & 0b1111]) << (4 * i))
        return ((output >> 11) | (output << (32 - 11))) & 0xFFFFFFFF
```

```
    def _decrypt_round(self, left_part, right_part, round_key):
        return left_part, right_part ^ self._f(left_part, round_key)

    def encrypt(self, plain_msg):
        def _encrypt_round(left_part, right_part, round_key):
            return right_part, left_part ^ self._f(right_part, round_key)

        left_part = plain_msg >> 32
        right_part = plain_msg & 0xFFFFFFFF
        for i in range(24):
            left_part, right_part = _encrypt_round(left_part, right_part, sel
f._subkeys[i % 8])
        for i in range(8):
            left_part, right_part = _encrypt_round(left_part, right_part, sel
f._subkeys[7 - i])
        return (left_part << 32) | right_part

    def decrypt(self, crypted_msg):
        def _decrypt_round(left_part, right_part, round_key):
            return right_part ^ self._f(left_part, round_key), left_part

        left_part = crypted_msg >> 32
        right_part = crypted_msg & 0xFFFFFFFF
        for i in range(8):
            left_part, right_part = _decrypt_round(left_part, right_part, sel
f._subkeys[i])
        for i in range(24):
            left_part, right_part = _decrypt_round(left_part, right_part, sel
f._subkeys[(7 - i) % 8])
        return (left_part << 32) | right_part


sbox = [numpy.random.permutation(l) for l in itertools.repeat(list(range(16))
, 8)]
sbox = (
    (4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3),
    (14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9),
    (5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11),
    (7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3),
    (6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2),
    (4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14),
    (13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12),
    (1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12),
    )

key = 183182793879123879127893789123798218793879782387932788723783298329823980
23031
```

```
text_short = input_for_cipher_short().encode().hex()
text_short = int(text_short, 16)

gost_short = GostCrypt(key, sbox)

encode_text_short = gost_short.encrypt(text_short)
decode_text_short = gost_short.decrypt(encode_text_short)
decode_text_short = bytes.fromhex(hex(decode_text_short)[2::]).decode('utf-
8')


text_long = input_for_cipher_long().encode().hex()
text_long = int(text_long, 16)

gost_long = GostCrypt(key, sbox)

encode_text_long = gost_long.encrypt(text_long)
decode_text_long = gost_long.decrypt(encode_text_long)
decode_text_long = bytes.fromhex(hex(decode_text_long)[2::]).decode('utf-8')


print(f'''
Гост 28147-89:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encode_text_short}

Расшифрованный текст:
{output_from_decrypted(decode_text_short)}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encode_text_long}

Расшифрованный текст:
{output_from_decrypted(decode_text_long)}
''')
```

Тестирование:

```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab05_14_gost89.py

Гост 28147-89:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
567540261451836962860566905831140964633059962168239725400849570714503615166865346335420862814981341444458694338340017794062562381816998355678753967115290745875600125177595186371409630906 82

Расшифрованный текст:
время,приливыиотливынеждутчеловека.
```

```
ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
266050417938476356087017230014873643206160447996114515129091774959747033405943890343897142869704450815393683605023684873888004909829680452295028718546978021649941177826880151732274533643864532320796630272450746837347684283063682350895571195422576014055891201234951917117574410777206443142305502703477508839984518986714583143400164257550139955951705291905260249349753577193635641327610319052903559952970011424379847993621475968548034595715017650775261104984599500375814441113013017875969917042030848511398372774550996207392191339181039778867511513245777658070846405184637307157764373019723472702707385419563060306821034731763551154503824714264812313316130706264074393244483028493575194280183695841632846694677441664704015987166764373865851781422124591521492824680316201781243695172538536956831291069312772130538659077026628200178989017827102455811284644197446147728227370823795695828448846841904902145096572681358073870106763470339835923246436026070889038200287505359250682058240850262033170450161709526251248889202783539456604116848059039836482794508570635524117505841700390724701975258248279406107572222767582208189240451665976152382603851822752929335645908960132784104671236312325136002226758406236192110306294400818405951215215100998698904096770829401135020739563435815427903679578218875067810914095989673219364329166908308361570998442869287890027766834584419183407652747042511617269389753273012998279245334616877159640347816015296464295681617647172047594328251693989022067560411048663904138961949717524467797892198348685209617903445304383592590876810894190958189473679242424610526622461775514367598680609644131641613792985187497438227966239919276204302869135347925244725013856160622834337882661360341824413825968528286788154642488982426559565499876930164748573179670457658054023039292389178257684534213982610204775980600094649100927108067273892323755053371019809720521867205334093895534822121423383888251650643168732003624170059793831256089491353050782028804944025904299184235348674984813250027251862086720688049102194441287254123131728573159600922255684796434774080500238892291475276783954053000145147885599279208235047400922773599948384461446382922212048102735317321789072559748723739437771744892787203600302455132063962866809231326655919329574793816888817133914475501975124214140801556180788374367162542394957389493526043951151164131904106002370372557788499423181314004878679937166122458155086667883866115359834887008407878066080617386495957544097625981039926098512363901470340589971200819260151401068029365898208739199000231395968707522505328350595047697302991894042689424537180277739107303643644783097835573010019102998769286433822677797732973441605094074880504351040539686018039879610535083300978658667265664391967425571351647822609084567301283329117759836853207234726828064964608886180317212877329147310950651510665931483604003665609249042923766617819843259861537060491859686395270315553393280970861384935830664056615086318518792532621575498933363420858503794211694857078464123907640827380597754238631523652222073583704208486423290328961434777281474628730883012418585005183526704604591722947806264958533139152742817759695979393904085889210131910527536391330455357053721771502167641015044650209047454483953632063217780184666438748654653608742685666423054367766118254669621964271957227771110446560009008260085554665704132125404748670093454836785013948460818856547409223615702492801811515171154353076248616600260838281861827510318136996678149054655293501452211334327394481409806145693576184764835508628555563178166027833341769254061164490913955537268308991417425837357068599913797622693757837464689346947323941
```

```
565580644564403759572656563966408160663598247766009685660723020552419971761400822686542373391732314068592807190033601276697644910836577636698226330648236553701481899448499076055004435636081490697301414470015727863530449650719384883208173939823273509148707571491685195572982448679697231841618128665142056093605453472388015139879689507614636267932199613494818601175694495106823965735679852248453340323322884737319847422163663683329727707507865218141639579913178721468797011563675058848404432420337436492844035343087300285655784488990284687493681659591330992349220652565296410779661349740175153061305288645417558786186533328282877556855948224417771287356978869466410912431116815452381165518582443064263928368461040647310768425435788168185304791521567591202904460425598689779839465289865075045548803625879099049880457756634096216287517583379850827114094281279725186959719047716798991463990201696001595394956661030446430788409672457741058597704420444123772447654933964197877421550863353732634744480181841548898737461517584844294120903204988150464053418061288143949058682199344504008861383378392402642556140683240706965932049456398214008601555594844595489325508401028823872678584029686001711761573768314347772017072852998336991250377282698317207452140810115763062272712262299394159864510307804974680262082930993792882105434472845097445767818724620636250333319570116562701500864340123631182252024978565591020490325434070533337461974698559903223030221449025366488862781604023577829136213595700371095616666191295280158427548094843204106432381377975693276853993148974688913227746868482199358865760126313652003609854882178324694
```

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходящийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпубликаций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.номожноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднукартину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотысячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблятьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеизменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиилибез.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностолькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,таккакэтопустоеместо.однаконекоторыефирмыибиржевидятсправедливымставитьстоимостьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосприятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.небольшинствунужнаценазатысячузнаковбезпробелов.

F: ПОТОЧНЫЕ ШИФРЫ

15.A5 /1

A5 — это поточный алгоритм шифрования, используемый для обеспечения конфиденциальности передаваемых данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи GSM (Groupe Spécial Mobile).

Шифр основан на побитовом сложении по модулю два (булева операция «исключающее или») генерируемой псевдослучайной последовательности и шифруемой информации. В A5 псевдослучайная последовательность реализуется на основе трёх линейных регистров сдвига с обратной связью. Регистры имеют длины 19, 22 и 23 бита соответственно. Сдвигами управляет специальная схема, организующая на каждом шаге смещение как минимум двух регистров, что приводит к их неравномерному движению. Последовательность формируется путём операции «исключающее или» над выходными битами регистров.

Код программы:

```
# -*- coding:utf-8 -*-
from base import alphabet, input_for_cipher_short, input_for_cipher_long, out
put_from_decrypted
import re
import copy
reg_x_length = 19
reg_y_length = 22
reg_z_length = 23
key_one = ""
reg_x = []
reg_y = []
reg_z = []


def loading_registers(key):
    i = 0
    while(i < reg_x_length):
        reg_x.insert(i, int(key[i]))
        i = i + 1
    j = 0
    p = reg_x_length
    while(j < reg_y_length):
        reg_y.insert(j, int(key[p]))
        p = p + 1
        j = j + 1
    k = reg_y_length + reg_x_length
    r = 0
    while(r < reg_z_length):
```

```
        reg_z.insert(r, int(key[k]))
        k = k + 1
        r = r + 1


def set_key(key):
    if(len(key) == 64 and re.match("^([01])+", key)):
        key_one = key
        loading_registers(key)
        return True
    return False


def get_key():
    return key_one


def to_binary(plain):
    s = ""
    i = 0
    for i in plain:
        binary = str(' '.join(format(ord(x), 'b') for x in i))
        j = len(binary)
        while(j < 12):
            binary = "0" + binary
            s = s + binary
            j = j + 1
    binary_values = []
    k = 0
    while(k < len(s)):
        binary_values.insert(k, int(s[k]))
        k = k + 1
    return binary_values


def get_majority(x, y, z):
    if(x + y + z > 1):
        return 1
    else:
        return 0


def get_keystream(length):
    reg_x_temp = copy.deepcopy(reg_x)
    reg_y_temp = copy.deepcopy(reg_y)
    reg_z_temp = copy.deepcopy(reg_z)
    keystream = []
    i = 0
    while i < length:
        majority = get_majority(reg_x_temp[8], reg_y_temp[10], reg_z_temp[10]
)
```

```
        if reg_x_temp[8] == majority:
            new = reg_x_temp[13] ^ reg_x_temp[16] ^ reg_x_temp[17] ^ reg_x_te
mp[18]
            reg_x_temp_two = copy.deepcopy(reg_x_temp)
            j = 1
            while(j < len(reg_x_temp)):
                reg_x_temp[j] = reg_x_temp_two[j-1]
                j = j + 1
            reg_x_temp[0] = new

        if reg_y_temp[10] == majority:
            new_one = reg_y_temp[20] ^ reg_y_temp[21]
            reg_y_temp_two = copy.deepcopy(reg_y_temp)
            k = 1
            while(k < len(reg_y_temp)):
                reg_y_temp[k] = reg_y_temp_two[k-1]
                k = k + 1
            reg_y_temp[0] = new_one

        if reg_z_temp[10] == majority:
            new_two = reg_z_temp[7] ^ reg_z_temp[20] ^ reg_z_temp[21] ^ reg_z
_temp[22]
            reg_z_temp_two = copy.deepcopy(reg_z_temp)
            m = 1
            while(m < len(reg_z_temp)):
                reg_z_temp[m] = reg_z_temp_two[m-1]
                m = m + 1
            reg_z_temp[0] = new_two

        keystream.insert(i, reg_x_temp[18] ^ reg_y_temp[21] ^ reg_z_temp[22])
        i = i + 1
    return keystream
def convert_binary_to_str(binary):
    s = ""
    length = len(binary) - 12
    i = 0
    while(i <= length):
        s = s + chr(int(binary[i:i+12], 2))
        i = i + 12
    return str(s)
def encrypt(plain):
    s = ""
    binary = to_binary(plain)
    keystream = get_keystream(len(binary))
    i = 0
    while(i < len(binary)):
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return s
```

```
def decrypt(cipher):
    s = ""
    binary = []
    keystream = get_keystream(len(cipher))
    i = 0
    while(i < len(cipher)):
        binary.insert(i, int(cipher[i]))
        s = s + str(binary[i] ^ keystream[i])
        i = i + 1
    return convert_binary_to_str(str(s))


def user_input_key():
    tha_key = str(input('Введите 64-bit ключ: '))
    if (len(tha_key) == 64 and re.match("^([01])+", tha_key)):
        return tha_key
    else:
        while(len(tha_key) != 64 and not re.match("^([01])+", tha_key)):
            if (len(tha_key) == 64 and re.match("^([01])+", tha_key)):
                return tha_key
            tha_key = str(input('Введите 64-bit ключ: '))
    return tha_key

# 0101001000011010110001110001100100101001000000110111111010110111


key = str(user_input_key())
set_key(key)


print(f'''
A5/1:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_short())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_short())))}

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
{encrypt(input_for_cipher_long())}

Расшифрованный текст:
{output_from_decrypted(decrypt(encrypt(
    input_for_cipher_long())))}
''')
```

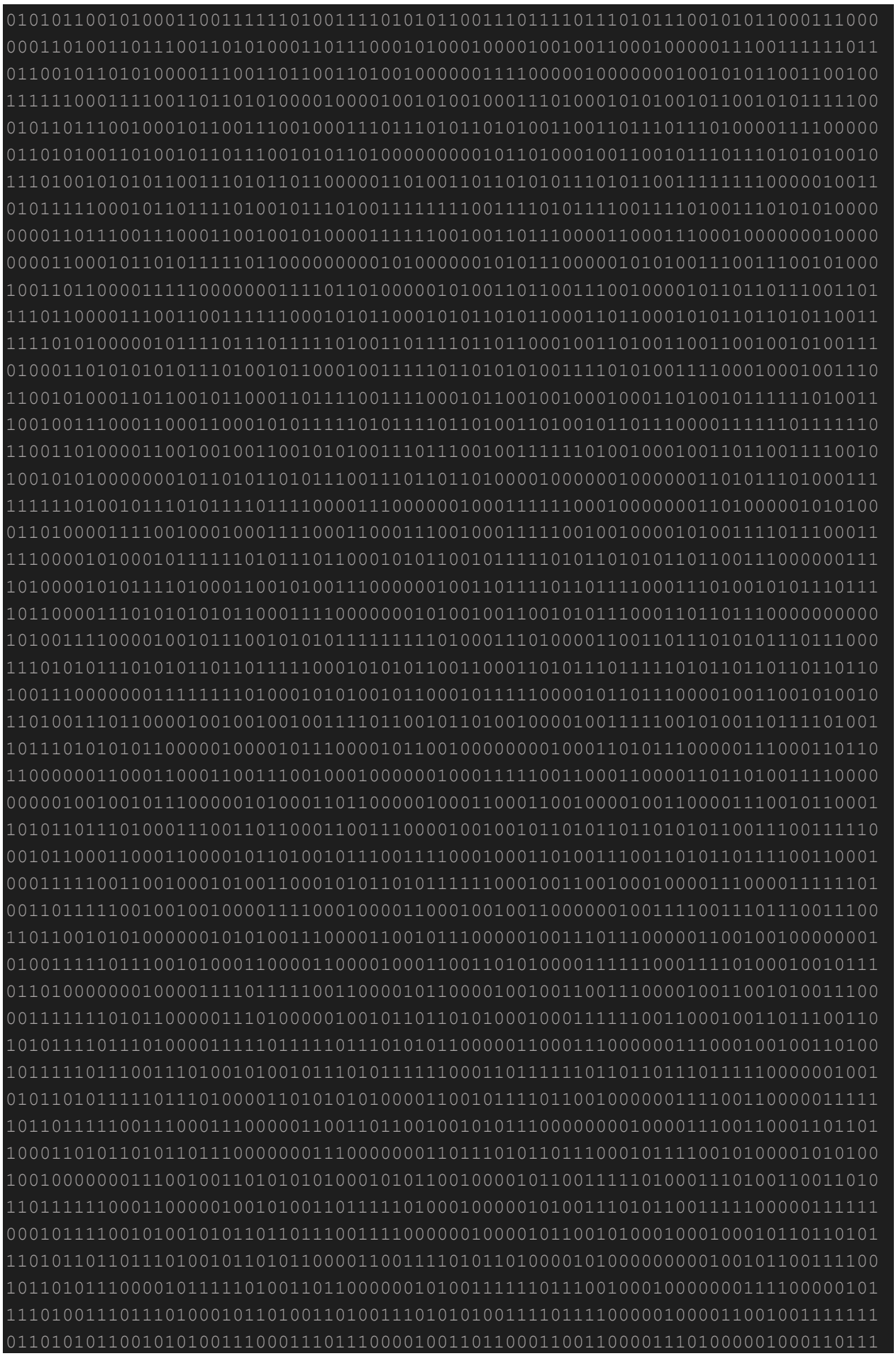
Тестирование:

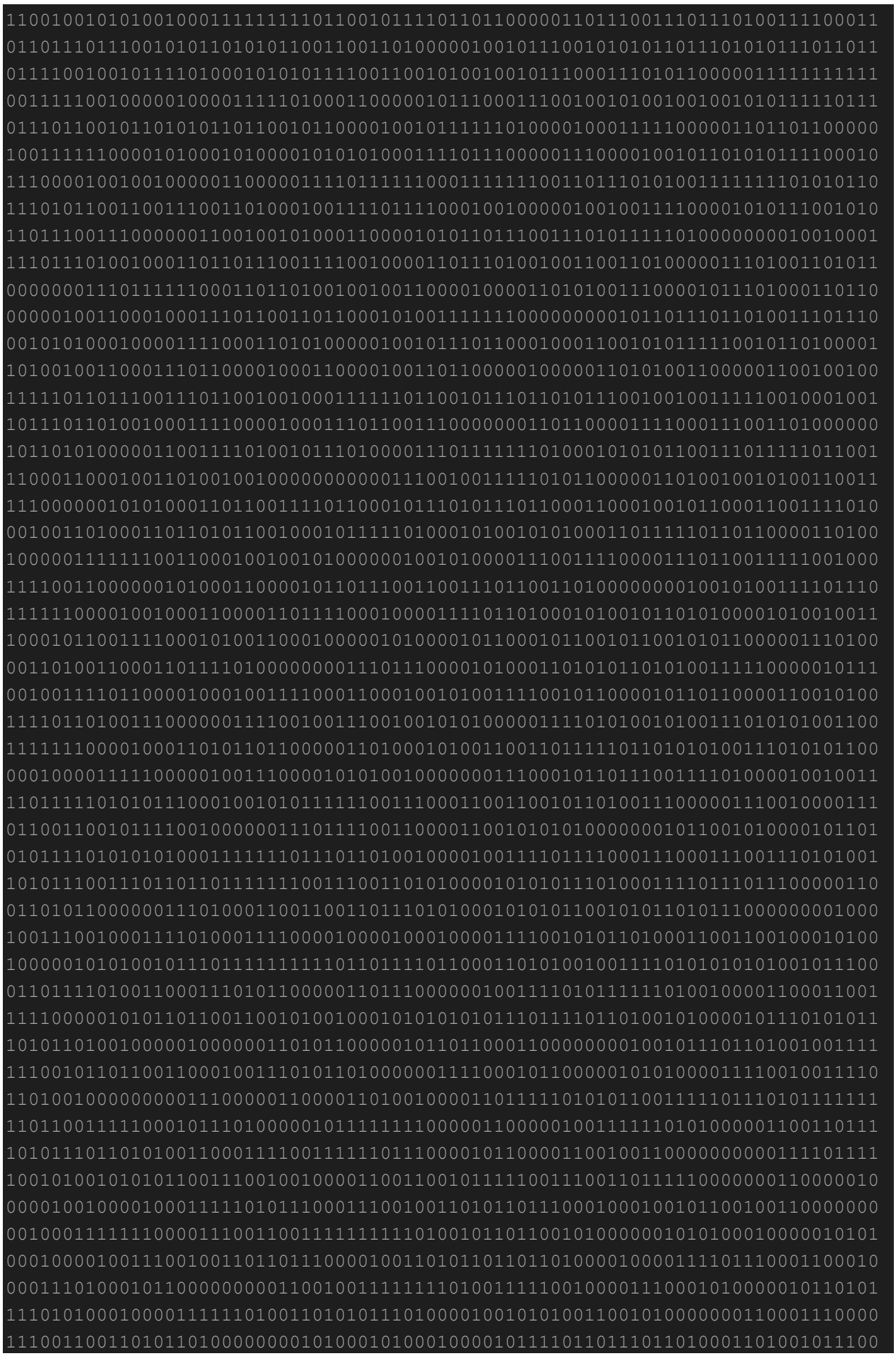
```
/bin/python3 /root/mospolytech-education-crypt-dev-2021-1/lab06_15_a51.py
Введите 64-bit ключ:
0101001000011010110001110001100100101001000000110111111010110111

A5/1:
КОРОТКИЙ ТЕКСТ:
Зашифрованный текст:
11011011101011101010000111010111001000001110010101111100100111011001001010110
00001011011100001111101111010001101101100110111000001000100110001011100111111
00110001110110110111111010110101110001111110100111001100000100110101001011110 1
100100011001101010010111010010100100010001000010101100101001011111011001010 11
11100010001010010100111111011000011000000101000010011010011000100001010101111
00110000100110110111000011011100100001000010001000011010111011101100111101011
111000

Расшифрованный текст:
время,приливыиотливынеждутчеловека.

ДЛИННЫЙ ТЕКСТ:
Зашифрованный текст:
11011011101011101101111111010000010100001110011001111100011011011001110110110
00001101011111100001101100101111101101100100111011111100100110011101100100001
10110000000100011011000011010101110011011110100000001100000100110101110011110 1
10101101000110010111111101001100010001111011001010001000100101111010100101011
01100010010110000100100000011000000111000101000000101010011010110001001010111
00111110001110110110111011010010011001001100011000010101011011100010100101011
11110010011011100000010100110110001111101101111001110111111010110100101110100
01010101110101101000111001100001101110110011100100100011111010100001011011010
00111100011011100001111110100111000101101000101001110010101100011011111110001
11100110001011000101100000010110110001001110010000111001001001011101101010100 
00110011101101110000000101001110011101010100111100001011000000100111101100000 1
00010001111111111111001101000011000000000110111000000000101101111000010110111 11
00011100100100100101100011101100010000000010101110011010101011111111101111001 0
11001110100110000100101111000001101010100100101101100001010101001000101011101
00010010000000000101001101011001011010111010111101101010110100000110111110001 1
00010110011000111011011110000000100000000110011010001111001100101111101111011 0
11111000110000100111000000110101110110111000101011001101001001011110011100001
11010111010110001001011011111101011100011110000101000000101000001100010001010 1
00110010000001001011110001111000000000101000000010110011101110001110110101010 0
11001100111001110111001010001010010100110110010110001100011011100011101101101
00000010101000111110100100110011001100110011011010000101011011010111100100100
00011000010011101100100111000010001001101111000011001001000000000010111101010
11100010001101011110010111001100011010110001111000110000000100011110101100011
00111001111110000000100111010010001010110011001010000100100101110100101010110 00
10011101100010001011111011111111011000001010010000011100000101000011000101101
10100100011100111111111010110000110001111100000111001010111011010111111101001
11011010001001111111000000101010101110000000110111110101100001101111011001101 00
00010011000000111110010101101011110000011101110110000000100111101001010000101 10
```





```
10010110111011010010001110110001111100100010011111110000000110000011000010111
00100010011100100010000001101110101111010110110011010011011101111000011111001
00011000001000010101010111001110011110101111010010110100011011101111111110010
00110001000100111001100111010111000110011010011011011011011010010011010011111
01111110000010100100111000010011010111001101000111111011000001111010101010100
00111010111111001100000000010101010110101100111001001110010011011000101011101
01110100000010010001100100111000111011100001111000100111111001110000001000010
00011100111000101000101000011100111011011100000010011010110011100111001111010
10111010011011010000010101010000111000100100101000000110110111110101011011001
01100001010011110000110011000111111010000100100111000111011001101010010010100
00011100001010010100111010101000001101110011000001011010001001100010100011000
11111101001011110100001001000100011100101101010001110111111110000110111001001
11111110101010111100000010110000111011100001010011010101000000010011001101101
10101000110110001011101000111100100000010001000011010111011111110010010001101
00011001111101101011100110000011101101110111010001011011001111100011011110001
00111011011100000001000011000010000011001001111010001111001001011011111111001
01101001010111110001000010101010001000000101000010110111011111111110110111001
10000000001011100000111100111111111111101011101110100100011110001100101010100
01101110001011011110010100110011011100110011110001100110010110010001000010111
01101111110000000010001101111000011000100111111100000100010010111001010000000
00010101000000101111011001011110110100010101100111001000100100000111111001000
10001111011001001000011110100001000001011001010111101100011011110010100011100
10011111000011001000111011000000110110101111111010111110101111111001100101101
11011001110111001101111000010000010101010001101000001111011110010110001000111
11001000111001000000110011101101010100111010100110001001101001010000010111111
00011100001010001010101011101001011000001010011001001111011110101010001101001
01100101101100101001100101000000000101110011101010101111011001101010010010000
10101100010010111010110110100110100000011010100000111100100000100001001011001
10110001110001001101100101111101100000010111111111110001001011001100100111111
01011110100001100100100110001110100110011001111111100101011111101100000110001
00001000100110100011110010101101010111000001001010001110000111001000011110111
00011101011001111010000101000101010110001001000111101111001000101100100101100
11011000011000000100011100010110011010011001100001111011010001010011000100101
11111100010010100000110110100111100011000111001011101010010101110001010110011
11111110100011011100001110101001011000110111001111011100000100100100110101001
11111010010001111000100000101101111101110010001010000011000111101001010101001
11001001000101111000100101100011001011101111001110100101010000001000000100010
10110101101110100101100000011111101011100010010111000111011000010101100100001
01011001100000011011110011001111000001001100111111100110111001000110111100110
11010110010011101010001101100111000101111010100000111001000100101101111000110
00100011011010001001100010000011010011101110101110110101110111010010011001100
01100100111111010101010110110 10
```

Расшифрованный текст:

вотпримерстатьинатысячусимволов.этодостаточномаленькийтекст,оптимальноподходя
щийдлякарточектовароввинтернетилимагазинахилидлянебольшихинформационныхпублик
аций.втакомтекстередкобываетболеедвухилитрёхабзацевиобычноодинподзаголовок.но
можноибезнего.натысячусимволоврекомендованоиспользоватьодинилидваключаиоднука
ртину.текстнатысячусимволовэтосколькопримернослов.статистикапоказывает,чтотыс
ячавключаетвсебястопятьдесятилидвестисловсреднейвеличины.но,еслизлоупотреблят

```
ьпредлогами,союзамиидругимичастямиречинаодинилидвасимвола,токоличествословнеи
зменновозрастает.вкопирайтерскойдеятельностипринятосчитатьтысячиспробеламиили
без.учетпробеловувеличиваетобъемтекстапримернонастоилидвестисимволовименностото
лькоразмыразделяемсловасвободнымпространством.считатьпробелызаказчикинелюбят,
таккакэтопустоеместо.однаконекоторыефирмыибиржвидятсправедливымставитьстоимо
стьзатысячусимволовспробелами,считаяпоследниеважнымэлементомкачественноговосп
риятия.согласитесь,читатьслитныйтекстбезединогопропуска,никтонебудет.нобольши
нствунужнаценазатысячузнаковбезпробелов.
```