Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

# ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

## ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ

студента 4 курса 431 группы

факультета компьютерных наук и информационных технологий

Мухи Семена Андреевича

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель _____ И.И. Слеповичев

подпись, дата

Саратов 2024

**Задание 1. Генерация псевдослучайных чисел**

Описание задания: создать программу, для генерации псевдослучайных величин. Входные параметры алгоритмы передаются программе в виде строки параметров, передаваемых через командную строку. Выходные значения записываются в файл, название которого указывается в строке параметров запуска программы.

**Алгоритм 1. Линейный конгруэнтный метод**

Описание алгоритма:

В основе линейного конгруэнтного метода лежит выбор четырех ключевых чисел:

- $m > 0$, модуль;
- $0 \leq a \leq m$, множитель;
- $0 \leq c \leq m$, приращение (инкремент);
- $0 \leq X_0 \leq m$, начальное значение.

Последовательность ПСЧ, получаемая по формуле:

$$x_{n+1} = \left(aX_n + c\right) mod\, m,\ n \geq 1$$

называется *линейной конгруэнтной последовательностью* (ЛКП). Ключом для неё служит $X_0$.

Параметры запуска программы:

/g:lc /n:10000 /i:6075;106;1283;7 /f:out.txt

Параметры алгоритма i:

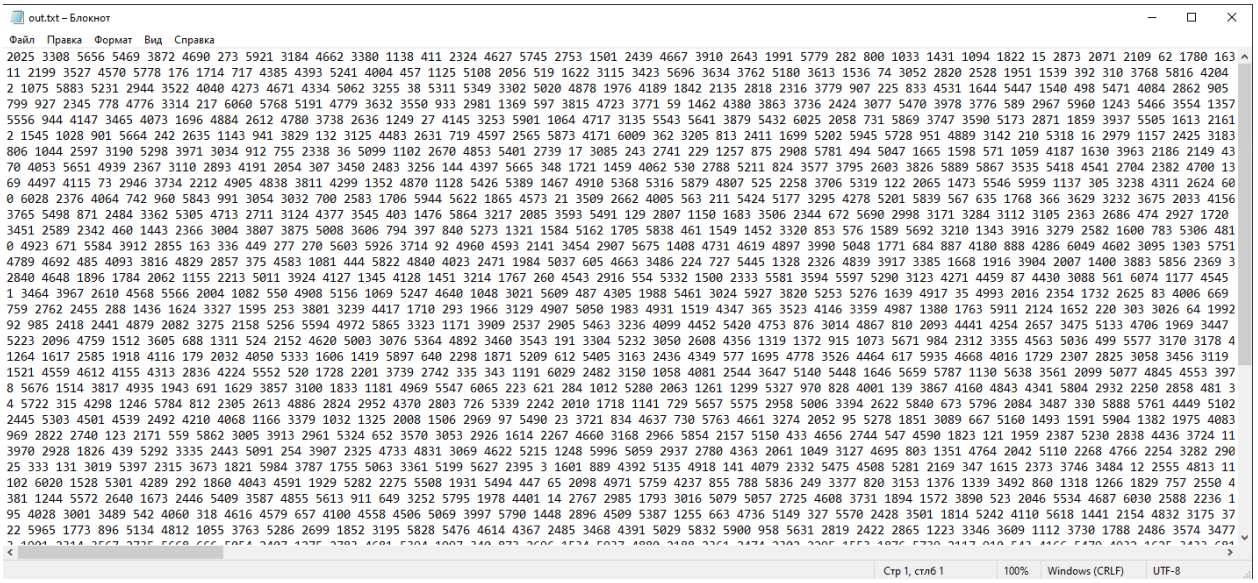
Модуль, множитель, приращение, начальное значение

Исходный текст алгоритма:

```
from progress.bar import IncrementalBar
def lc(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
    res = []
    if parameters_[0]:
                    m,  a,  c,  x  =  int(parameters_[0]),
```

```
int(parameters_[1]), int(parameters_[2]), int(parameters_[3])
        else:
            m, a, c, x = 6075, 106, 1283, 7
        for i in range(n_):
            x = (a * x + c) % m
            res.append(x)
            bar.next()
        bar.finish()
        return res
```

Результат работы программы:




**Алгоритм 2. Аддитивный метод**

Описание алгоритма:

Каждое следующее значение вычисляется по рекуррентной формуле:

$$x_{n+1} = \left(x_{n-k} + x_{n-j}\right) mod\, m,\ \ j > k \geq 1.$$

Числа $k,\ j$ – целые числа, которые называются запаздываниями, $m$ – это модуль, $n$ – длина вектора, который подается на вход, $x_0$ берется из вектора начальных значений.

Параметры запуска программы:

/g:add /n:10000 /i:128;9;49 /f:out.txt

Параметры алгоритма i:

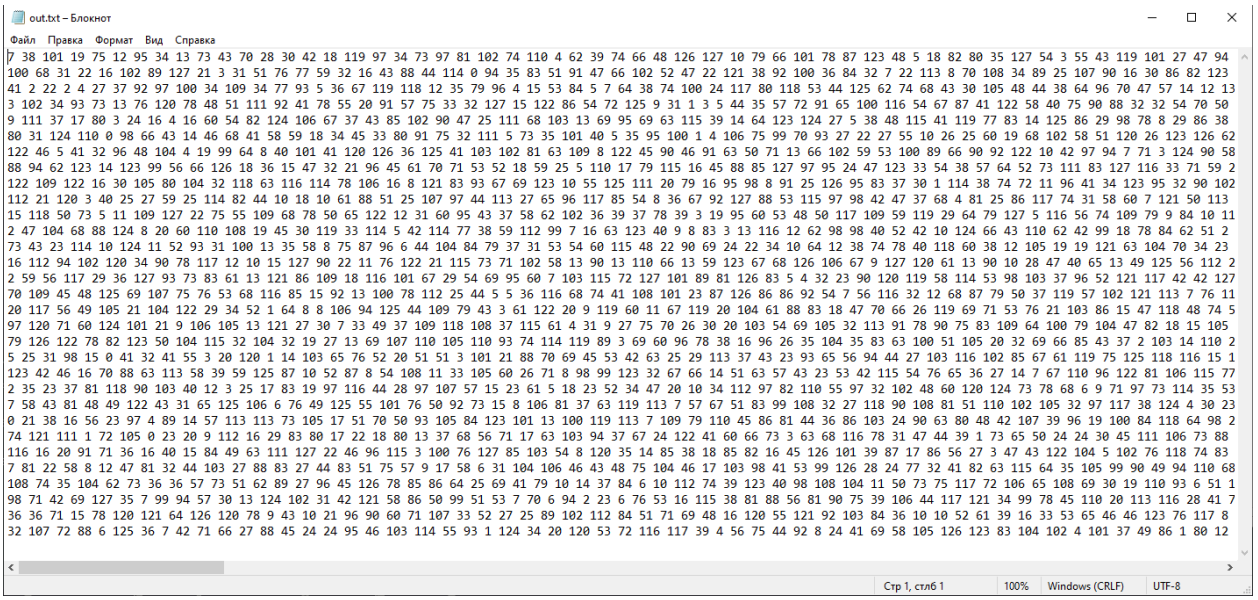
Модуль, младший индекс, старший индекс, последовательность

начальных значений.

Исходный текст алгоритма:

```
def add(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
    res = []
    if parameters_[0]:
        xs = list(int(i) for i in parameters_) + [31, 93, 35, 97, 35, 58, 85, 32, 93, 16, 61, 78, 46, 67, 5, 66, 5, 97, 9, 83, 30, 51, 85, 84,89, 30, 27, 53, 72, 32, 92, 13, 93, 5, 1, 97, 95, 24, 53, 28, 75, 4, 62, 4, 21, 75, 50, 6, 8, 3, 14, 62, 49, 95, 8]
    else:
        xs = [1024, 9, 49, 31, 93, 35, 97, 35, 58, 85, 32, 93, 16, 61, 78, 46, 67, 5, 66, 5, 97, 9, 83, 30, 51, 85, 84, 89, 30, 27, 53, 72, 32, 92, 13, 93, 5, 1, 97, 95, 24, 53, 28, 75, 4, 62, 4, 21, 75, 50, 6, 8, 3, 14, 62, 49, 95, 8]
    m = xs.pop(0)
    k = xs.pop(0)
    j = xs.pop(0)
    base_lan = len(xs)
    for n in range(base_lan, n_ + base_lan):
        x = (xs[n - k] + xs[n - j]) % m
        xs.append(x)
        res.append(x)
        bar.next()
    bar.finish()
    return res
```

Результат работы программы:




**Алгоритм 3. Пятипараметрический метод**

Описание алгоритма:

Данный метод является частным случаем РСЛОС, использует характеристический многочлен из 5 членов. Генерирует последовательности -битовых двоичных чисел в соответствии с рекуррентной формулой:

$$X_{n+p} = X_{n+q_1} + X_{n+q_2} + X_{n+q_3} + X_n,\ n = 1,\ 2,\ 3, \ldots$$

Где $X_i$ это биты.

Параметры $q_1$, $q_2$ и $q_3$ являются показателями степени ненулевых членов характеристического полинома, а $p$ – размер начального вектора.

Необходимо чтобы удовлетворялось условие, где

$$p > q_1 > q_2 > q_3 > 0$$

Y имеет такое же значение, как и в РСЛОС.

Параметры запуска программы:

/g:5p /n:10000 /f:out.txt /i:89;20;40;69;10;12

Параметры алгоритма i:

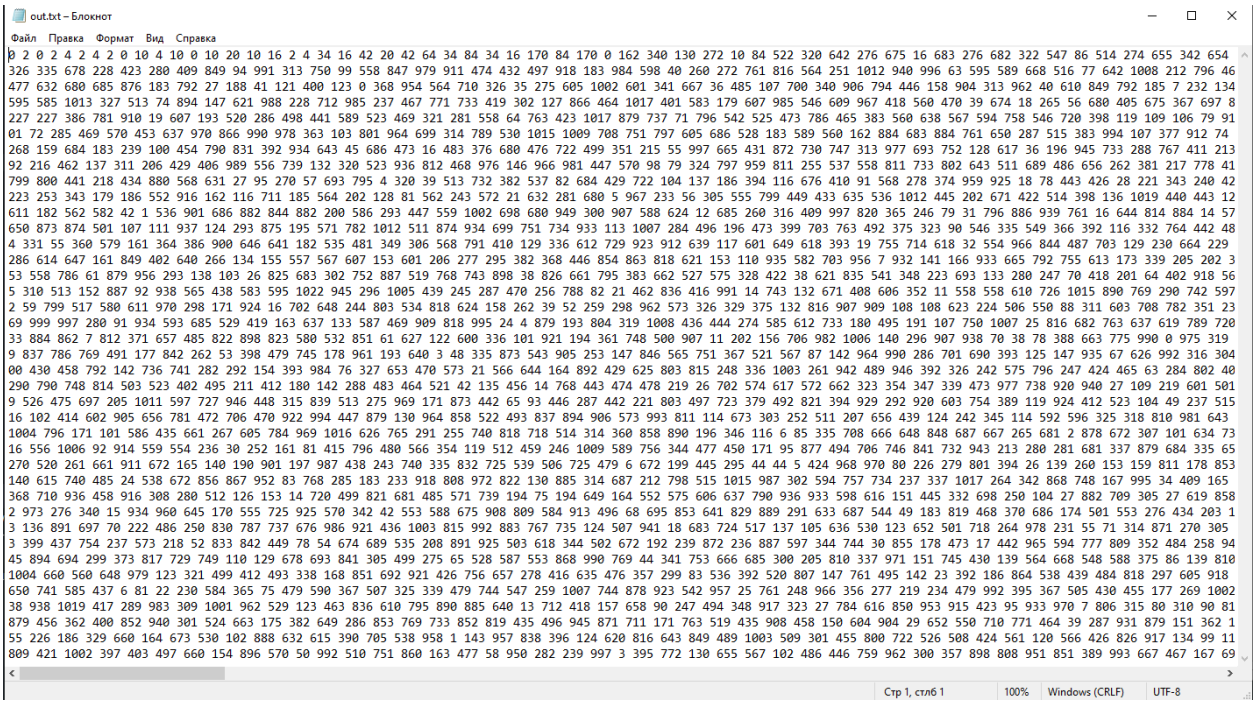
p, q1, q2, q3, w, начальное значение

Исходный текст алгоритма:

```
def p5(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
```

```
        res = []
        if parameters_[0]:
                p, q1, q2, q3, w, key = int(parameters_[0]),
int(parameters_[1]), int(parameters_[2]), int(parameters_[3]), \
                                                int(parameters_[4]),
str(parameters_[5])
        else:
            p, q1, q2, q3, w, key = 89, 20, 40, 69, 10, 12
        while len(key) < p:
            key = '0' + key
        counter = 0
        for i in range(n_):
            x_bin = ''
            for j in range(w):
                 x = int(key[counter + q1]) + int(key[counter +
q2]) + int(key[counter + q3]) + int(key[counter])
                counter += 1
                x = x % 2
                key += str(x)
                x_bin += str(x)
            res.append(int(x_bin, 2))
            bar.next()
        bar.finish()
        return res
```

Результат работы программы:




**Алгоритм 4. Регистр сдвига с обратной связью (РСЛОС)**

Описание алгоритма:

Для натурального числа $p$ и параметров $a_1, a_2, \ldots, a_{p-1}$, которые принимают значения либо 0, либо 1, работает рекуррентная формула:

$$x_{n+p} = a_{p-1}x_{n+p-1} + a_{p-2}x_{n+p-2} + \ldots + x_n \ (mod\ 2)$$

Наименьшее положительное целое число $N$, такое, что $X_{n+N} = X_n$ для всех значений $n$ называют периодом последовательности. Эту последовательность называют М-последовательностью. Период данной последовательности составляет $(2^p - 1)$.

Значения $x_i$ и $a_i$ лежать в диапазоне $x_1, \ldots, x_p,\ a_1, \ldots,\ a_p \in \{0.1\}$.

Индексы $1 \leq j_1 < \ldots < j_m \leq p$, означают, что только $a_{j_i} = 1$, остальные коэффициенты равно 0.

Параметры запуска программы:

/g:lfsr /n:10000 /f:out.txt /i:1010110101010;541;10

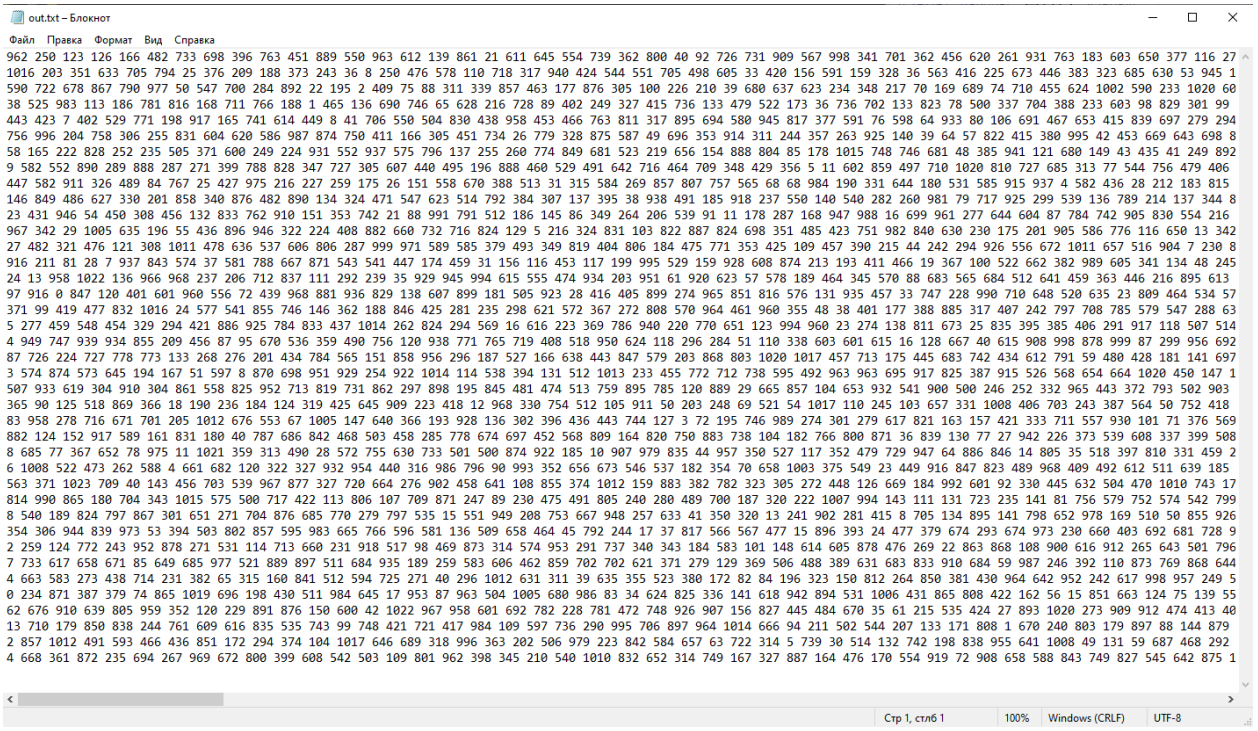
Параметры алгоритма i:

Двоичное представление вектора коэффициентов, начальное значение регистра.

Исходный текст алгоритма:

```
def lfsr(flag, n_, parameters_):
    if flag:
        bar = IncrementalBar('Выполнение:', max=n_)
    taps = []
    if parameters_[0]:
        for i, j in enumerate(str(parameters_[0])[::-1]):
            if j == '1':
                taps.append(i)
        state = int(parameters_[1])
        w = int(parameters_[2])
    else:
        for i, j in enumerate(str(101101010)[::-1]):
            if j == '1':
                taps.append(i)
        state = 541
        w = 10
    feedback = 0
    res = []
    for i in range(n_):
        x_bin = ''
        for j in range(w):
            for tap in taps:
                feedback ^= (state >> tap) & 1
                state = ((state << 1) | feedback) & ((1 <<
max(taps) + 1) - 1)
            x_bin += str(state >> 1 & 1)
        res.append((int(x_bin, 2)))
        if flag:
            bar.next()
    if flag:
        bar.finish()
    return res
    return res
```

Результат работы программы:




**Алгоритм 5. Нелинейная комбинация РСЛОС. Генератор Геффа**

Описание алгоритма:

Генератор Геффа является примером нелинейной комбинацией РСЛОС. В этом генераторе используются три РСЛОС, объединённые нелинейным образом. Длины этих регистров $L_1, L_2, L_3$ - попарно простые числа. Нелинейная функция генератора:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$$

Параметры запуска программы:

/g:nfsr /n:10000 /f:out.txt /i:1010110101010;1010110101010; 1010110101010;541;993;117;10

Параметры алгоритма i:

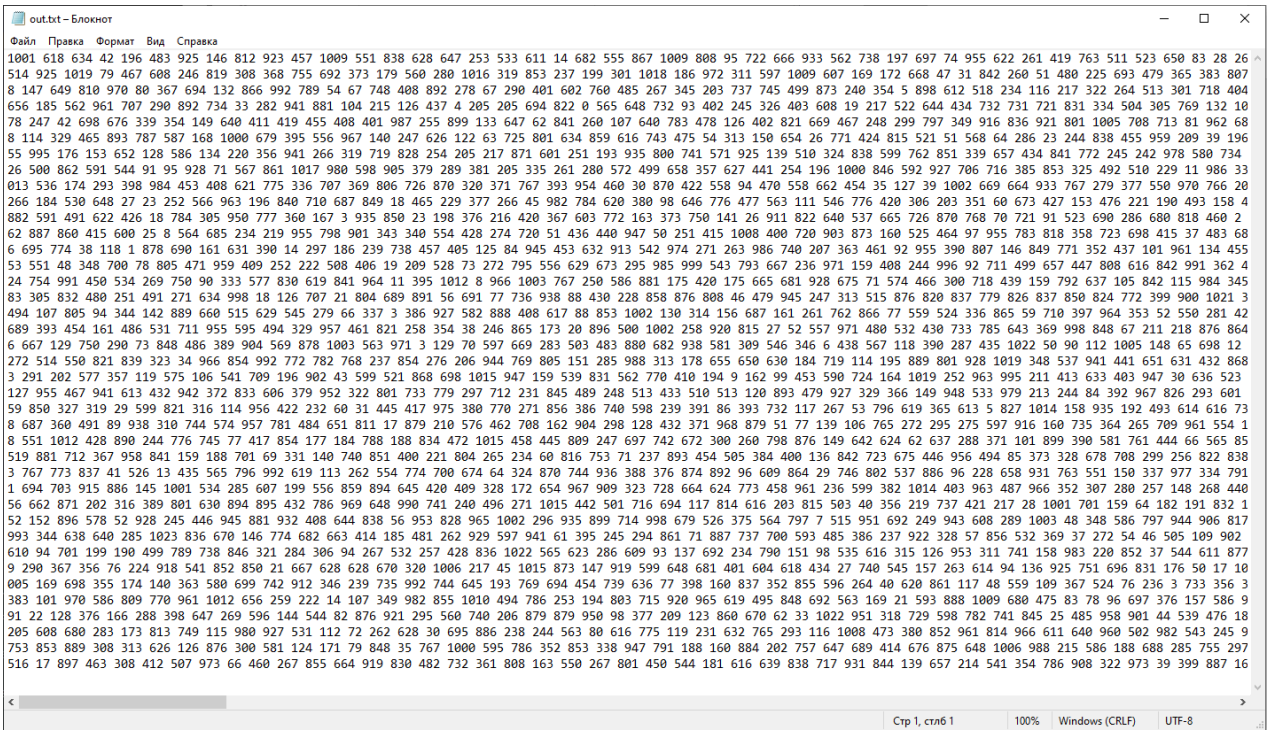
Параметры – двоичное представление векторов коэффициентов для R1, R2, R3, w, x1, x2, x3. w – длина слова, x1, x2, x3 – десятичное представление начальных состояний регистров R1, R2, R3.

Исходный текст алгоритма:

```
def nfsr(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
```

```
        result = []
        if parameters_[0]:
            w = parameters_[6]
                    x1  =  lfsr(False,  n_,  [parameters_[0],
parameters_[3], w])
                    x2  =  lfsr(False,  n_,  [parameters_[1],
parameters_[4], w])
                    x3  =  lfsr(False,  n_,  [parameters_[2],
parameters_[5], w])
        else:
            w = 10
            x1 = lfsr(n_, [1010110101010, 541, w])
            x2 = lfsr(n_, [1010110101010, 993, w])
            x3 = lfsr(n_, [1010110101010, 117, w])
        for i in range(0, n_):
            x1Element = x1[i]
            x2Element = x2[i]
            x3Element = x3[i]
             resultTemp = (x1Element & x2Element) ^ (x2Element &
x3Element) ^ x3Element
            result.append(resultTemp)
            bar.next()
        bar.finish()
        return result
```

Результат работы программы:




**Алгоритм 6. Вихрь Мерсенна**

Описание алгоритма:

Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых w-битных чисел в соответствии с рекуррентной формулой:

$$x_{n+p} = x_{n+q} \oplus \left(X^{l}_{n+1}\right) A \ (n = 0,\ 1,\ 2,\ \ldots),$$

где $p,\ q,\ r$ – целые константы;

$p$ – степень рекуррентности, $1 \leq q \leq p$;

$X_n$ – $w$-битное двоичное целое число;

$\left(X^{l}_{n+1}\right)$ – двоичное целое число, полученное конкатенацией чисел $X^{r}_{n}$ и $X^{l}_{n+1}$когда первые $(w - r)$ битов взяты из $X_n$, а последние $r$ битов из $X_{n+1}$ в том же порядке;

$A$ – матрица размера $w \times w$ состоящая из нулей и единиц, определенная

посредством $a$.

$XA$ – произведение, при вычислении которого сначала выполняют операцию $X >> 1$ (сдвига битов на одну позицию вправо), если последний бит $X$ равен 0, а затем, когда последний бит $X = 1$, то вычисляют $XA = (X >> 1) \oplus a$.

Параметры задаются изначально:

$w$ – размер слова (разрядность значений, которыми оперирует алгоритм)

$r$: $r \leq w$ – позиция разделения

$p, q$: $0 < q \leq p$ *–два положительных числа*

$a, b, c$: $0 \leq a, b, c < 2^w$-*w-разрядные неотрицательные числа*

$u, s, t, l$: $0 \leq u, s, t, l \leq w$ *– коэффициенты*

$x_0$, ..., $x_{p-1}$ *– начальные значения вектора*

Параметры запуска программы:

/g:mt /n:10000 /f:out.txt /i:624

Параметры алгоритма i:

Модуль, начальное значение x

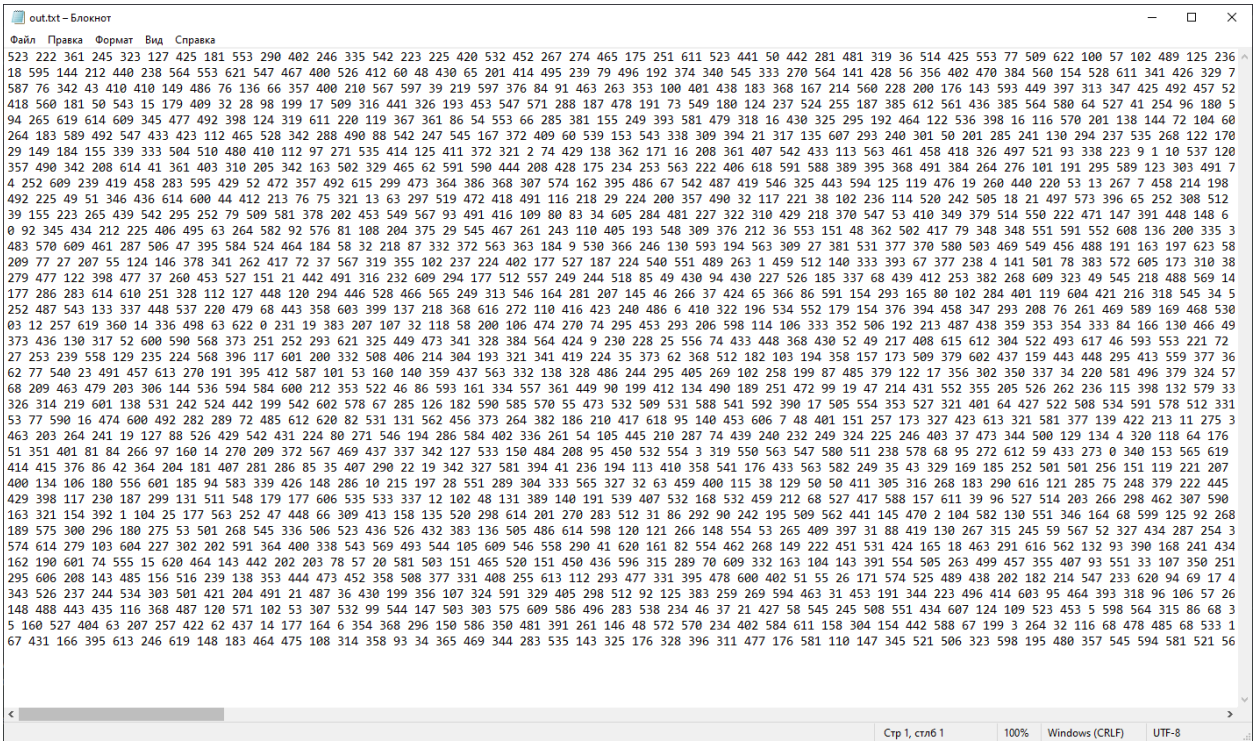
Исходный текст алгоритма:

```
def mt(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
    w = 32
    r = 31
    q = 397
    a = 2567483615
    u = 11
    s = 7
    t = 15
    l = 18
    b = 2636928640
    c = 4022730752
    if parameters_[0]:
        p = int(parameters_.pop(0))
    else:
        p = 624
```

```
        if parameters_[0]:
            x = parameters_
        else:
            x = [5, 28, 14, 27, 18, 17, 23, 1, 24, 17, 5, 11, 1,
1, 19, 13, 21, 7, 2, 11, 8, 23, 21, 15, 24, 4, 7, 11, 29, 7, 24,
15, 13, 9, 26, 3, 12, 9, 8, 17, 2, 2, 28, 6, 30, 14, 29, 21, 13,
1, 27, 9, 18, 26, 8, 14, 22, 15, 9, 7, 21, 10, 12, 6, 23, 17,
13, 18, 3, 29, 29, 17, 5, 14, 18, 18, 17, 12, 5, 3, 18, 26, 29,
29, 7, 17, 1, 23, 16, 9, 26, 28, 4, 21, 6, 30, 29, 15, 14, 26,
24, 30, 23, 26, 22, 22, 26, 9, 2, 16, 11, 16, 30, 3, 7, 30, 8,
11, 9, 10, 25, 12, 1, 22, 11, 5, 22, 12, 24, 18, 17, 6, 10, 15,
21, 24, 26, 12, 13, 4, 19, 26, 26, 22, 15, 10, 1, 18, 25, 28, 1,
24, 18, 27, 3, 5, 15, 27, 21, 17, 5, 29, 16, 28, 30, 10, 26, 6,
22, 6, 4, 4, 8, 5, 2, 4, 17, 22, 5, 12, 15, 11, 8, 7, 5, 5, 8,
18, 23, 28, 19, 2, 18, 6, 2, 3, 9, 17, 9, 4, 29, 6, 29, 17, 25,
11, 18, 28, 12, 6, 13, 8, 14, 14, 7, 13, 9, 22, 28, 20, 30, 3,
8, 1, 28, 10, 28, 7, 12, 26, 14, 27, 18, 30, 7, 18, 2, 30, 13,
12, 11, 17, 9, 24, 23, 10, 18, 4, 15, 29, 3, 19, 15, 24, 13, 15,
7, 12, 3, 7, 21, 17, 24, 3, 14, 22, 9, 29, 2, 7, 27, 29, 18, 26,
4, 12, 6, 25, 21, 8, 20, 11, 10, 23, 8, 4, 26, 28, 12, 19, 11,
13, 1, 3, 22, 12, 16, 11, 6, 26, 28, 17, 25, 29, 5, 12, 27, 25,
11, 7, 13, 5, 27, 12, 19, 25, 11, 5, 3, 5, 9, 26, 28, 25, 18,
18, 22, 16, 17, 29, 2, 20, 2, 7, 2, 26, 29, 6, 6, 23, 8, 20, 6,
26, 24, 28, 22, 15, 7, 28, 26, 7, 24, 21, 28, 16, 27, 8, 2, 3,
19, 23, 6, 20, 19, 27, 16, 16, 1, 20, 10, 8, 8, 8, 28, 21, 8,
11, 4, 13, 29, 29, 8, 24, 22, 3, 2, 26, 13, 19, 8, 17, 25, 6, 2,
7, 4, 20, 24, 26, 2, 23, 9, 15, 22, 19, 20, 24, 29, 2, 29, 24,
3, 28, 30, 2, 22, 28, 21, 28, 9, 12, 30, 18, 13, 2, 9, 17, 20,
10, 24, 30, 20, 23, 6, 30, 21, 8, 26, 13, 30, 9, 30, 1, 14, 19,
16, 6, 18, 9, 15, 1, 27, 4, 12, 4, 26, 6, 24, 19, 24, 4, 15, 6,
13, 10, 24, 2, 29, 5, 12, 24, 14, 24, 11, 1, 23, 24, 12, 2, 2,
18, 27, 30, 11, 26, 28, 20, 20, 8, 11, 23, 4, 26, 19, 17, 21,
11, 29, 11, 30, 2, 9, 12, 17, 18, 18, 13, 14, 12, 19, 20, 7, 15,
2, 17, 15, 26, 12, 24, 22, 3, 4, 22, 16, 9, 12, 16, 13, 30, 14,
24, 1, 10, 21, 16, 6, 1, 30, 27, 19, 25, 27, 7, 12, 17, 24, 29,
12, 20, 4, 21, 12, 16, 13, 21, 23, 29, 2, 29, 21, 12, 13, 23,
```

```
12, 22, 16, 12, 19, 22, 6, 20, 11, 28, 16, 7, 26, 14, 17, 17, 4,
22, 29, 6, 27, 14, 16, 28, 18, 11, 25, 2, 13, 27, 14, 23, 27,
14, 30, 21, 6, 6, 4, 12, 15, 17, 27, 3, 6, 5, 2, 19, 9, 12, 24,
20, 11, 21, 13, 8, 26, 16, 18, 1]
        res = []
        value1 = ''
        value2 = ''
        for i in range(w - r):
            value1 += '1'
            value2 += '0'
        for i in range(r):
            value1 += '0'
            value2 += '1'
        value1Int = int(value1, 2)
        value2Int = int(value2, 2)
        n = int(n_)
        for i in range(n + 1500):
            t12 = int(x[i]) & value1Int
            t13 = int((x[i + 1])) & value2Int
            Y = t12 | t13
            if (Y % 2 != 0):
                valuex = (int(x[i + q]) % 2 ** w) ^ (Y >> 1) ^ a
            else:
                valuex = (int(x[i + q]) % 2 ** w) ^ (Y >> 1) ^ 0
            Y = valuex
            Y = (Y ^ (Y >> u))
            Y = Y ^ ((Y << s) & b)
            Y = Y ^ ((Y << t) & c)
            Z = (Y ^ (Y >> l))
            x.append(valuex)
            res.append(Z % p)
            bar.next()
        bar.finish()
        return res
```

Результат работы программы:




**Алгоритм 7. RC4**

Описание алгоритма:

**На вход:**

n – количество необходимых сгенерированных чисел

w – количество бит, используемых для генерации числа

K – массив ключа длины 256, которые состоит из чисел от 0 до 255, которые перемешаны любым способом.

1. Инициализация $S_i$, $i = 0, 1, ..., 255$.

Начальное значение состояния заполняется на основе ключа $K_0$, ..., $K_{255}$

a) $for\ i = 0\ 255 : S_i = i;$

b) $j = 0;$

c) $for\ i = 0\ to\ 255: j = (j + S_i + K_i)\ mod\ 256;\ Swap(Si, Sj)$

2. $i = 0,\ j = 0.$

3. Итерация алгоритма:

a) $i = (i + 1)\ mod\ 256;$

b) $j = (j + S_i) \bmod 256$;

c) $Swap(S_i, S_j)$;

d) $t = (S_i + S_j) \bmod 256$;

e) $K = S_t$;

Параметры запуска программы:

/g:rc4 /n:10000 /f:out.txt

/i:73;25;169;67;200;69;83;93;19;100;141;85;207;66;71;236;194;239;167;32;101;135;213;35;89;112;188;178;82;33;206;54;249;51;255;102;164;155;133;46;16;231;152;42;122;15;41;14;208;244;230;6;8;245;217;124;227;185;184;248;37;59;31;191;120;111;26;253;140;63;125;242;3;136;27;36;186;95;220;94;243;49;70;150;79;118;117;176;172;247;24;65;241;238;174;55;114;21;2;129;162;17;210;254;22;60;62;251;91;215;0;109;223;156;97;11;127;123;130;250;192;1;138;52;113;160;181;229;105;47;44;40;180;116;168;153;154;201;72;234;128;224;5;161;197;134;132;190;177;29;56;12;77;50;58;74;131;146;246;68;166;96;216;219;212;13;115;211;157;144;203;20;232;9;45;34;23;151;195;237;196;80;57;7;205;43;182;193;106;87;104;119;38;218;148;48;187;221;226;159;145;202;110;228;173;209;10;108;75;84;139;53;61;18;103;183;98;179;165;81;126;137;171;170;252;147;78;214;163;158;233;149;142;175;121;76;90;4;92;199;30;107;88;99;189;222;225;143;235;39;240;204;28;86;198

Параметры алгоритма i:

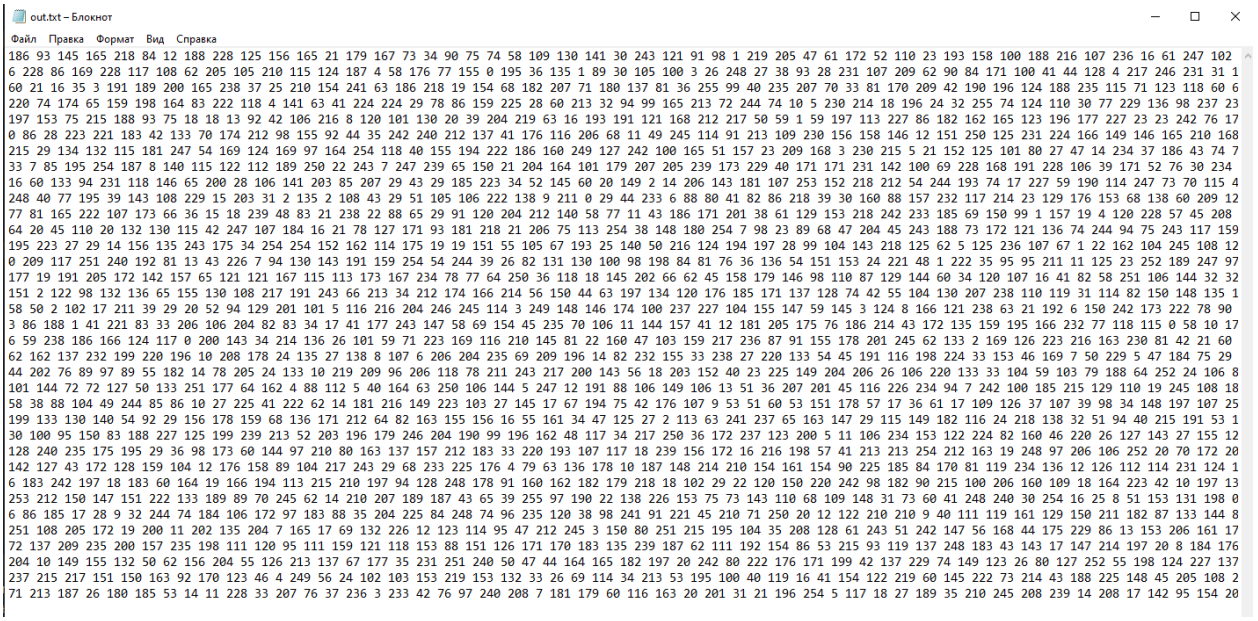
256 начальных значений

Исходный текст алгоритма:

```
def rc4(n_, parameters_):
    res = []
    bar = IncrementalBar('Выполнение:', max=n_)
    length = 256
    if parameters_[0]:
        secretKey = parameters_
    else:
```

```
        secretKey = ['1', '2', '3', '4', '5', '6', '7',
'31', '48', '178', '46', '91', '30', '15', '132', '154',
'108', '222', '245', '211', '103']
    sBox = [0] * length
    key = [0] * length
    for i in range(length):
        sBox[i] = i
    for i in range(length):
        key[i] = int(secretKey[i % len(secretKey)])
    j = 0
    for i in range(length):
        j = (j + sBox[i] + key[i]) % length
        sBox[i], sBox[j] = sBox[j], sBox[i]
    j = 0
    for i in range(n_, 0, -1):
        i = (i + 1) % length
        j = (j + sBox[i]) % length
        sBox[i], sBox[j] = sBox[j], sBox[i]
        t = (sBox[i] + sBox[j]) % length
        res.append(sBox[t])
        bar.next()
    bar.finish()
    return res
```

Результат работы программы:




**Алгоритм 8. RSA**

Описание алгоритма:

Основные параметры:

**На вход:**

c – количество необходимых сгенерированных чисел

n – модуль

e – степень

$x_0$- начальное значение

w – количество генерируемых бит за шаг

l – длина выходной двоичной последовательности

1. Выбрать случайное целое $x_0$.

2.count = (c * l) / w

$For\ i\ =\ 1\ to$ count $do$

a. $x_i \leftarrow x_{x-1}^{e}\ mod\ n$

b. В поток добавляем w бит из сгенерированного числа

3. “Нарезаем” последовательность по l бит $z_1, z_2, ..., z_l$

4. Переводим последовательность $z_1, z_2, ..., z_l$ в десятичное число.

Параметры запуска программы:

/g:rsa /n:10000 /f:out.txt /i:100151;224951;287;22528;22

Параметры алгоритма i:

Модуль n, число e, w, начальное значение x.

e удовлетворяет условиям: $1 < e < (p-1)(q-1)$, НОД$(e, (p-1)(q-1)) = 1$, где $p*q=n$.

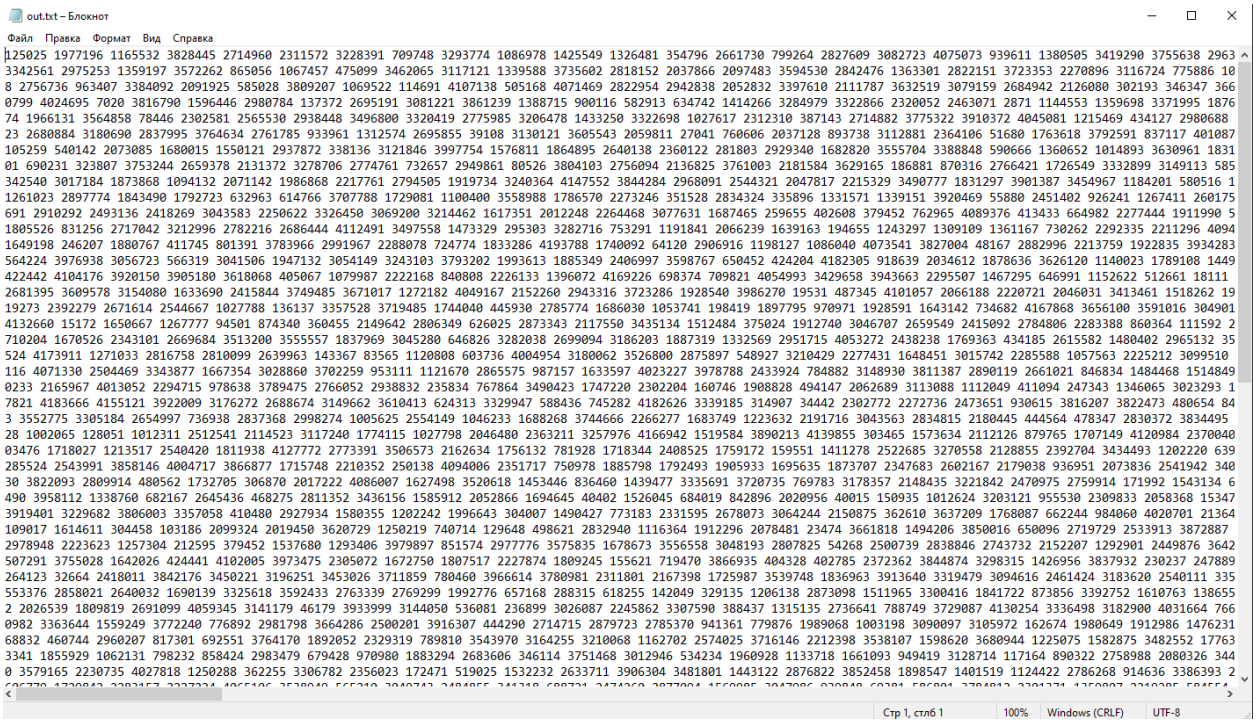
x из интервала [1,n]

w – длина слова.

Исходный текст алгоритма:

```
def rsa(n_, parameters_):
                p,   q,   e,   x,   l   =   int(parameters_[0]),
int(parameters_[1]), int(parameters_[2]), int(parameters_[3]),\
                        int(parameters_[4])
        res = []
        bar = IncrementalBar('Выполнение:', max=n_)
        n = p * q
        f = (p - 1) * (q - 1)
        for i in range(n_):
            counter = l - 1
            seqElem = 0
            for j in range(l):
                x = x ** e % n
                bit = x & 1
                seqElem = seqElem | (bit << counter)
                counter -= 1
            res.append(seqElem)
            bar.next()
        bar.finish()
        return res
```

Результат работы программы:




**Алгоритм 9. Алгоритм Блюм–Блюма–Шуба (BBS)**

Описание алгоритма:

Основные параметры:

$n$- модуль

$x_0$- начальное значение

$l$- длина выходной последовательности

**На входе:** Модуль n, начальное значение $x_0$, длина выходной последовательности $l$.

**На выходе:** Последовательность псевдослучайных бит $z_1, z_2, ..., z_l$, переведенных в десятичное число.

Модуль $m = p * q$ является произведение двух больших простых чисел $p$ и $q$.

1. Вычислим $x_0 = x^2 \, mod \, n$, которое будет начальным вектором.

2. $For \; i$=1 $to \; l \; do$

1. $x_{i+1} \leftarrow x_i^2 mod \, n$

2. $z_i \leftarrow$ последний значащий бит $x_i$

3. Вернуть $z_1, z_2, ..., z_l$.

4. Перевести последовательность бит $z_1, z_2, ..., z_l$ в десятичное число.

Параметры запуска программы:

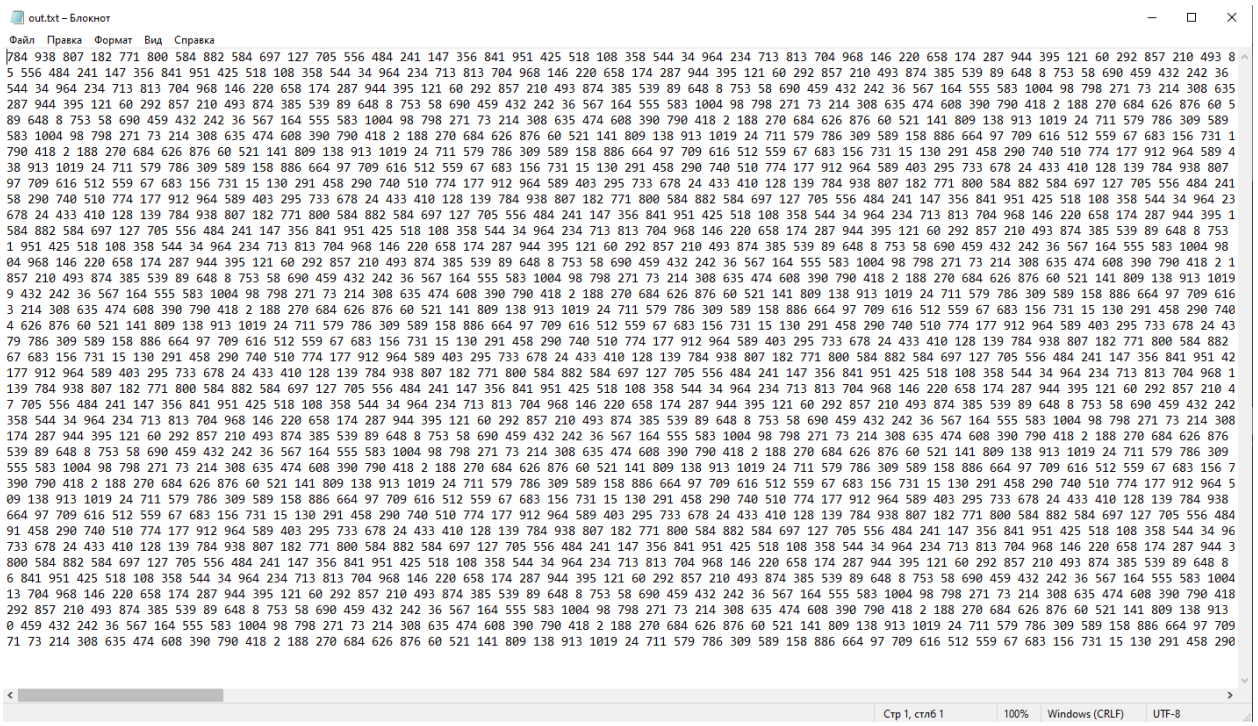
/g:bbs /n:10000 /i:7 /f:out.txt

Начальное значение x (взаимно простое с n).

Исходный текст алгоритма:

```
def bbs(n_, parameters_):
    bar = IncrementalBar('Выполнение:', max=n_)
    res = []
    if parameters_[0]:
        x = int(parameters_[0])
    else:
        x = 7
    n, w = 50621, 10    #16637
    for i in range(n_):
        x_bin = ''
        for j in range(w):
            x = (x * x) % n
            x_bin += str(x % 2)
        res.append(int(x_bin, 2))
        bar.next()
    bar.finish()
    return res
```

Результат работы программы:


out.txt – Блокнот
Файл Правка Формат Вид Справка
784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 8
5 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36
544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635
287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 5
89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589
583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 1
790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 4
38 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807
97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241
58 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 23
678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 1
584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753
1 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98
04 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 1
857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019
9 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616
3 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740
4 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 43
79 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882
67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 42
177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 1
139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 4
7 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242
358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308
174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876
539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309
555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 7
390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 5
09 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938
664 97 709 616 512 559 67 683 156 731 15 130 291 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484
91 458 290 740 510 774 177 912 964 589 403 295 733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 96
733 678 24 433 410 128 139 784 938 807 182 771 800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 3
800 584 882 584 697 127 705 556 484 241 147 356 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8
6 841 951 425 518 108 358 544 34 964 234 713 813 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004
13 704 968 146 220 658 174 287 944 395 121 60 292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418
292 857 210 493 874 385 539 89 648 8 753 58 690 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913
0 459 432 242 36 567 164 555 583 1004 98 798 271 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709
71 73 214 308 635 474 608 390 790 418 2 188 270 684 626 876 60 521 141 809 138 913 1019 24 711 579 786 309 589 158 886 664 97 709 616 512 559 67 683 156 731 15 130 291 458 290
Стр 1, стлб 1
100%
Windows (CRLF)
UTF-8