

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Математический факультет
Кафедра функционального анализа

Отчет по дисциплине:
«Программирование криптографических алгоритмов»

Направление 02.04.01 Математика и компьютерные науки

Преподаватель	_____	к.ф.-м.н.	М.Г. Завгородний
	<i>подпись</i>		
Обучающаяся	_____		С.Д. Бабошин
	<i>подпись</i>		

Воронеж 2021

Содержание

1	Постановка задачи	3
2	Используемые технологии	4
3	Структура программы	5
4	Описание используемых алгоритмов	6
4.1	Сложение	6
4.2	Вычитание	6
4.3	Умножение	6
4.4	Деление с остатком	7
4.5	Возведение в степень	8
4.6	Вычисление корня из натурального числа	8
4.7	НОД	8
4.8	Сложение в кольце вычетов	9
4.9	Вычитание в кольце вычетов	9
4.10	Умножение в кольце вычетов	9
4.11	Обратные элементы в кольце вычетов	9
4.12	Возведение в степень в кольце вычетов	9
5	Блок-схемы алгоритмов	10
6	Примеры работы программы	22
7	Список литературы	24
8	Исходный код	25
8.1	bigint.py	25
8.2	tests.py	34

1 Постановка задачи

1. Составьте алгоритм (в виде блок-схемы) и напишите (на любом языке программирования) соответствующую ему программу, позволяющую выполнять арифметические операции (сложение, вычитание, умножение и деление) над длинными целыми числами;
2. Составьте алгоритм и напишите соответствующую ему программу, позволяющую возводить натуральное число в натуральную степень;
3. Составьте алгоритм и напишите соответствующую ему программу, позволяющую вычислять целую часть корня произвольной степени $m > 0$ из натурального числа;
4. Используя один из предложенных выше алгоритмов, составьте блок-схему и напишите соответствующую ей программу, позволяющую вычислять наибольший общий делитель двух больших натуральных чисел;
5. Составьте алгоритм и напишите соответствующую ему программу, позволяющую выполнять операции сложения, вычитания и умножения в кольце вычетов;
6. Составьте алгоритм и напишите соответствующую ему программу, позволяющую находить элементы, обратные к элементам, взаимно простым с модулем кольца;
7. Составьте алгоритм и напишите соответствующую ему программу, позволяющую возводить в натуральную степень элементы кольца вычетов.

2 Используемые технологии

Программа написана на языке программирования Python 3.9. Plusом данного решения стало то, что ЯП Python поддерживает работу с большими целыми числами и это позволило легко написать тесты для моей программы. Программа использует только стандартную библиотеку Python, установка сторонних зависимостей не требуется.

3 Структура программы

Вся программа состоит из трёх файлов:

1. Основная программа (*bigint.py*). В данном файле содержится класс *BigInt*, который позволяет совершать арифметические операции с длинными целыми числами. Для удобства работы с данным классом были перегружены основные арифметические операторы (такие как «+», «-», «*» и пр.) и это позволило работать с объектами данного класса как с обычными числами.
2. Библиотека функций для длинной арифметики (*long_math.py*). В данном файле содержатся функции, которые работают с длинными числами и вызываются из класса *BigInt*. Блок-схемы данных функций будут представлены ниже.
3. Тесты работы программы (*tests.py*). В данном файле содержатся юнит-тесты со следующим принципом работы:
 - (a) Случайно выбираем 2 числа в промежутке от -10^{30} до 10^{30} ;
 - (b) Преобразуем их в тип *BigInt*. После этого у нас будет 2 пары одинаковых чисел. Одна пара типа *int* из стандартной библиотеки, а вторая типа *BigInt*;
 - (c) Производим арифметические действия на обеих парах чисел и сравниваем получившиеся результаты. Если результаты отличаются, то выводим ошибку;
 - (d) Выполняем предыдущие пункты 100000 раз.

4 Описание используемых алгоритмов

4.1 Сложение

Сложение реализовано в функции *l_add*. Для сложения используется алгоритм описанный в [1].

4.2 Вычитание

Вычитание реализовано в функции *l_sub*. Для вычитания используется алгоритм описанный в [1].

4.3 Умножение

Умножение реализовано в функции *l_mul*. Для умножения используется исправленный алгоритм умножения из [1]. Были внесены следующие изменения (синие строки были изменены, красные удалены, а зелёные добавлены):

1. Вводим числа x и y в строковые переменные $s1$ и $s2$ соответственно.
2. Определяем длины $l1$ и $l2$ строк $s1$ и $s2$ соответственно.
3. Полагаем $m = \max\{l1, l2\}$
4. Полагаем $k = (m - 1) / 4 + 1$
5. Полагаем $n = 4 * k$
6. Дописываем $n - l1$ нулей в начало строки $s1$ и $n - l2$ нулей в начало строки $s2$
7. Полагаем $osn = 10^4$, $st = '0'$, $n1 = n$
8. Цикл при изменении переменной j от 1 до k выполняем:
 - (a) Полагаем $n1 = n$ и $w = 0$
 - (b) Из строки $s2$ считываем 4 символа, начиная с позиции $n1 - 3$, преобразуем их в числовой формат и присваиваем целочисленной переменной b .

- (c) Полагаем $n2 = n$, $w = 0$, $s3 = 0$
- (d) Цикл при изменении переменной i от 1 до k выполняем:
- i. Из строки $s1$ считываем 4 символа, начиная с позиции $n - 3$, преобразуем их в числовой формат и присваиваем целочисленной переменной a .
 - ii. Находим величину $c = a * b + w$
 - iii. Если $c < osn$, то $z = c$, $w = 0$, иначе $z = c \% osn$, $w = c / osn$
 - iv. Преобразуем число z в строковый формат и присваиваем строковой переменной s .
 - v. Если длина l строки s меньше 4, то дописываем 4 - l нулей в начало строки s .
 - vi. В начало строки $s3$ дописываем четыре символа строки s .
 - vii. Полагаем $n = n - 4$ и $i = i + 1$
 - viii. Полагаем $s3 = s + s3$, $n2 = n2 - 4$, $i = i + 1$
- (e) Если после выполнения i -цикла имеем $w \neq 0$, то число w преобразуем в строковый формат и полученную строку добавляем в начало строки $s3$.
- (f) Дописываем $4(j-1)$ нулей в конец строки $s3$.
- (g) Используя алгоритм сложения, складываем числа, записанные в строках st и $s3$; результат сложения записывем в строковую переменную st .
- (h) Полагаем $n1 = n1 - 4$ и $j = j + 1$

4.4 Деление с остатком

Деление с остатком реализовано в функции l_divmod . В качестве алгоритма используется деление в столбик с небольшими модификациями для ускорения работы и сокращения количества итераций.

4.5 Возведение в степень

Возведение в степень реализовано в функции *l_pow*. Для возведения числа в степень используется алгоритм описанный в [1].

4.6 Вычисление корня из натурального числа

Вычисление корня из натурального числа реализовано в функции *l_root*. Для вычисления корня используется алгоритм описанный в [1] и [2].

4.7 НОД

В программе реализовано 3 алгоритма вычисления НОД:

1. Расширенный алгоритм Евклида [1] (реализован в методе *BigInt.gcd*)
2. Расширенный бинарный алгоритм [1] (реализован в методе *BigInt.bin_gcd*).

Однако в предложенной реализации содержится ошибка, которую мне не удалось исправить. В связи с этим в данном алгоритме некорректно вычисляются коэффициенты линейного представления НОД. Пример: $(927, 238) = 1$, при этом $u = 257, v = 100$, что, очевидно, неверно. Правильные значения u и v это 19 и -74 соответственно: $927 * 19 + 238 * (-74) = 1$.

3. Алгоритм LSBGCD [1] (реализован в методе *BigInt.lsbgcd*). В исходный алгоритм, предложенный в [1] были внесены следующие изменения:

- Первое и самое главное изменение содержится в пункте алгоритма 3.1: "Найти число n такое, что $2^n y \leq x < 2^{n+1} y$. В [1] предлагается **на каждой** итерации основного цикла полагать $n = \log_2(10) \cdot L_{10}(y)$. Однако, посмотрев как меняется n , я пришёл к выводу, что начинать поиск с этого значения на **каждой** итерации цикла не оптимально, т.к. n изменяется на малую величину (максимальное увиденное мной изменение переменной n равнялось 4), а следовательно, разумно будет поменять алгоритм следующим образом:

- **До** основного цикла полагаем $n = \log_2(10) \cdot L_{10}(y)$.

- На каждой итерации цикла осуществляем поиск нового n , начиная с предыдущего n .

В моём случае данная оптимизация ускорила алгоритм практически в 8 раз!

- Арифметические выражения и цикл поиска значения n были слегка переработаны таким образом, чтобы 2^n можно было вынести в переменную.

4.8 Сложение в кольце вычетов

Сложение реализовано в методе *BigInt.ring_add*. Для сложения используется алгоритм описанный в [1].

4.9 Вычитание в кольце вычетов

Вычитание реализовано в методе *BigInt.ring_sub*. Для вычитания используется алгоритм описанный в [1].

4.10 Умножение в кольце вычетов

Умножение реализовано в методе *BigInt.ring_mul*. Для умножения используется алгоритм описанный в [1].

4.11 Обратные элементы в кольце вычетов

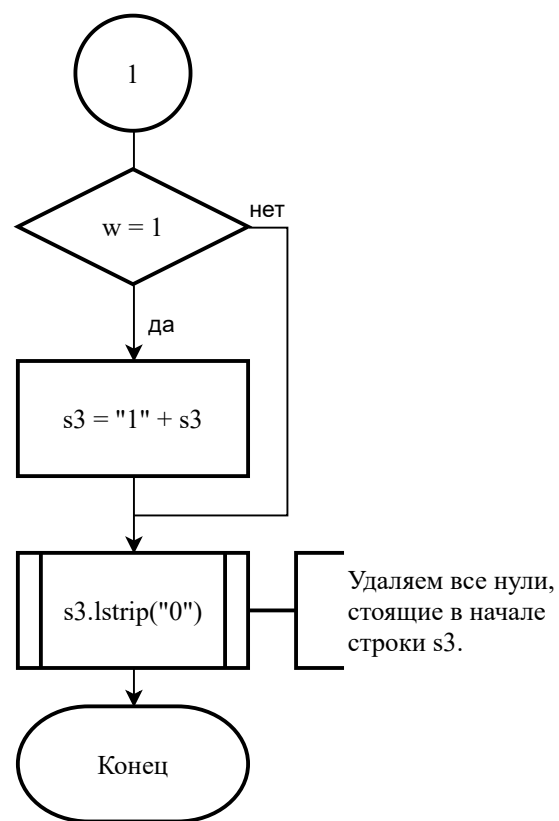
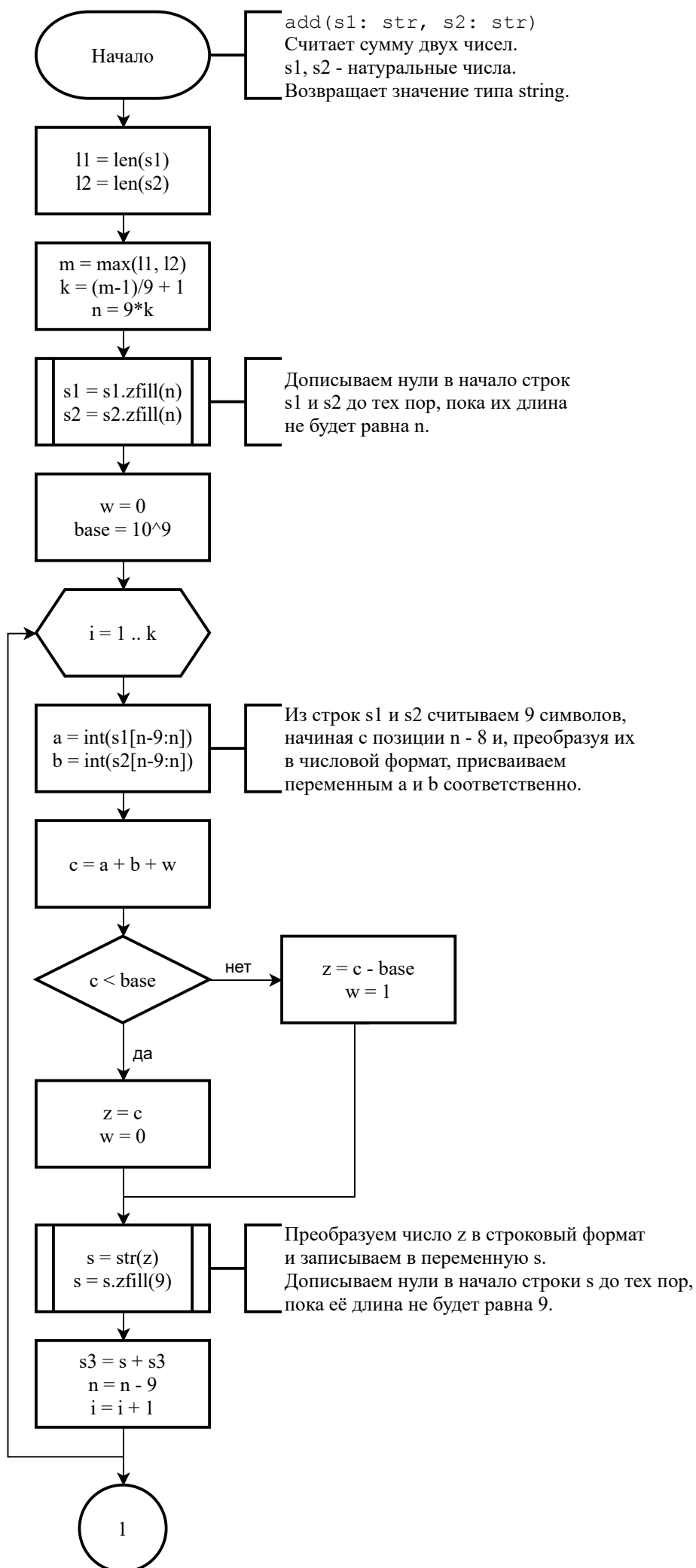
Вычисление обратных элементов реализовано в методе *BigInt.ring_inv*. Для вычисления обратных элементов используется алгоритм описанный в [1].

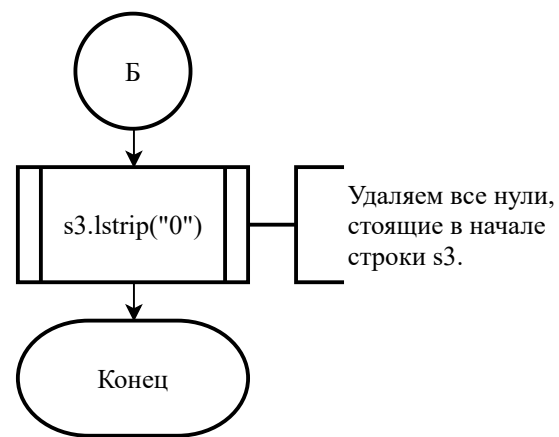
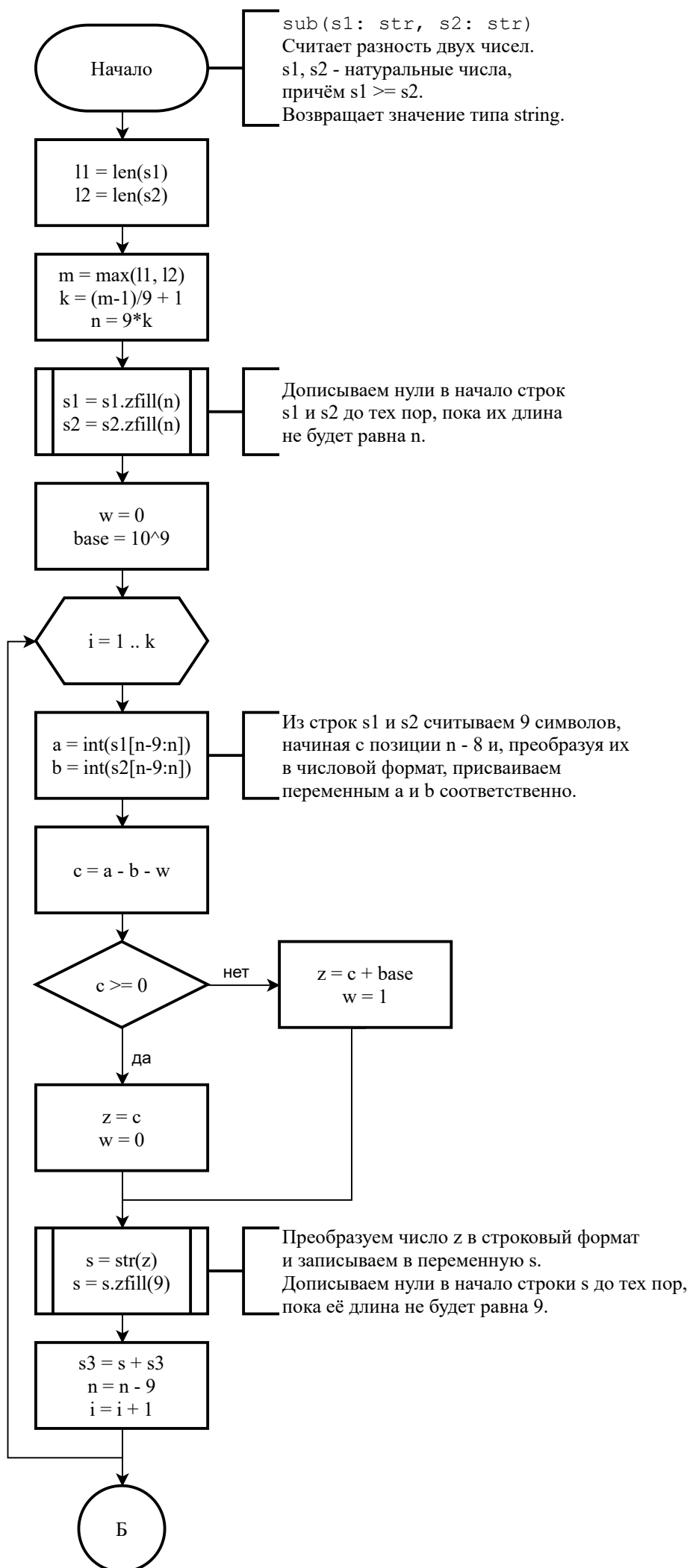
4.12 Возведение в степень в кольце вычетов

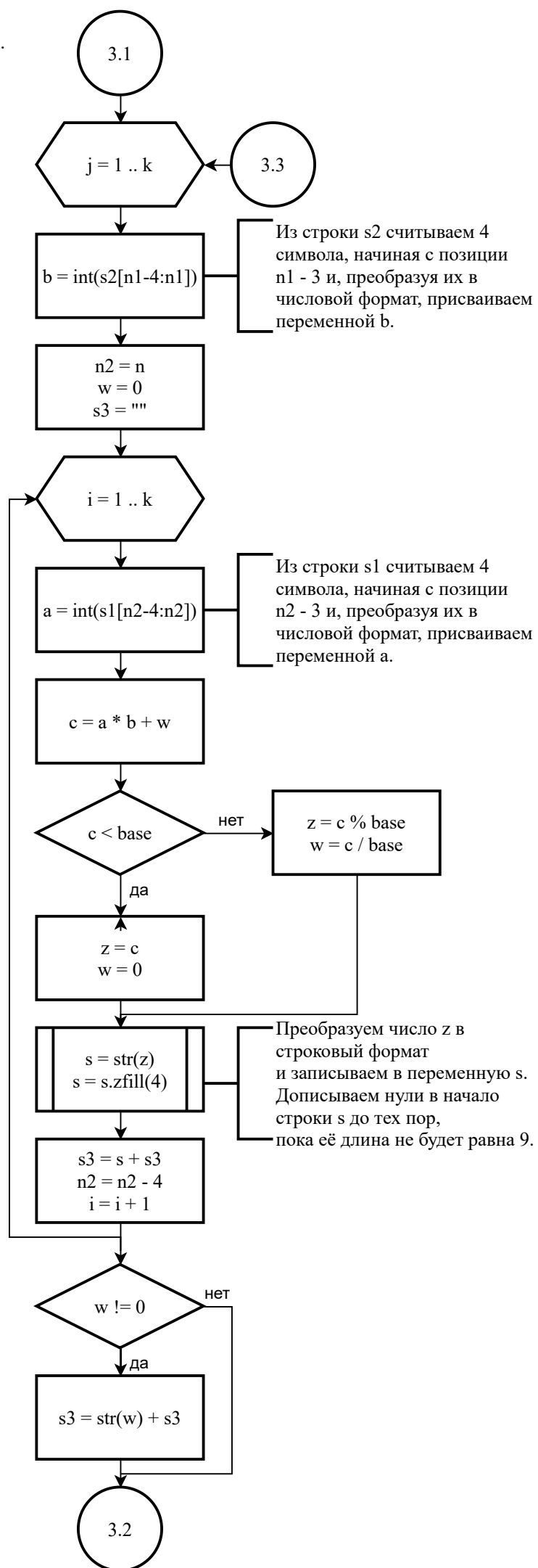
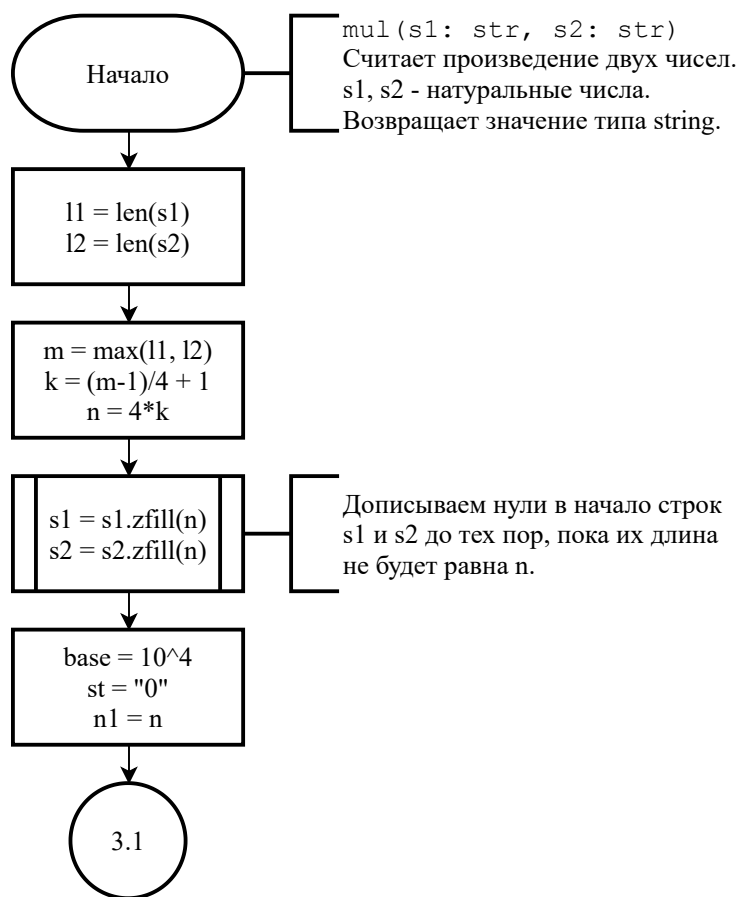
Возведение в степень реализовано в методе *BigInt.ring_pow*. Для возведения в степень используется алгоритм описанный в [1].

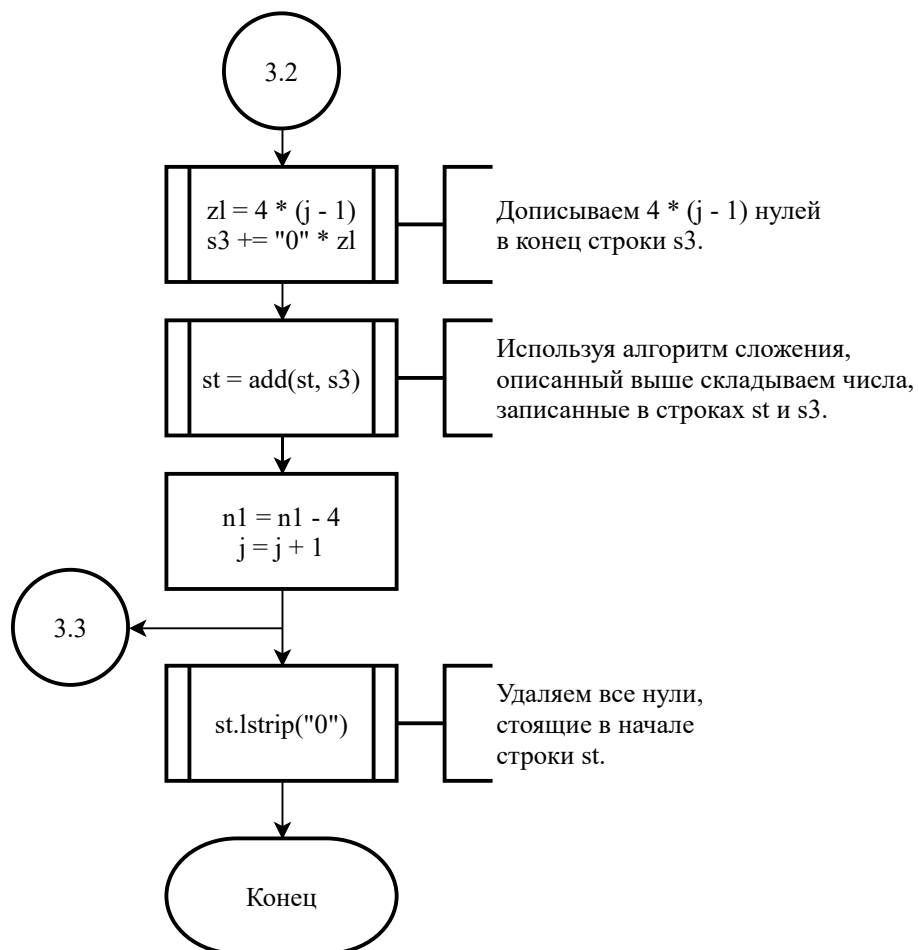
5 Блок-схемы алгоритмов

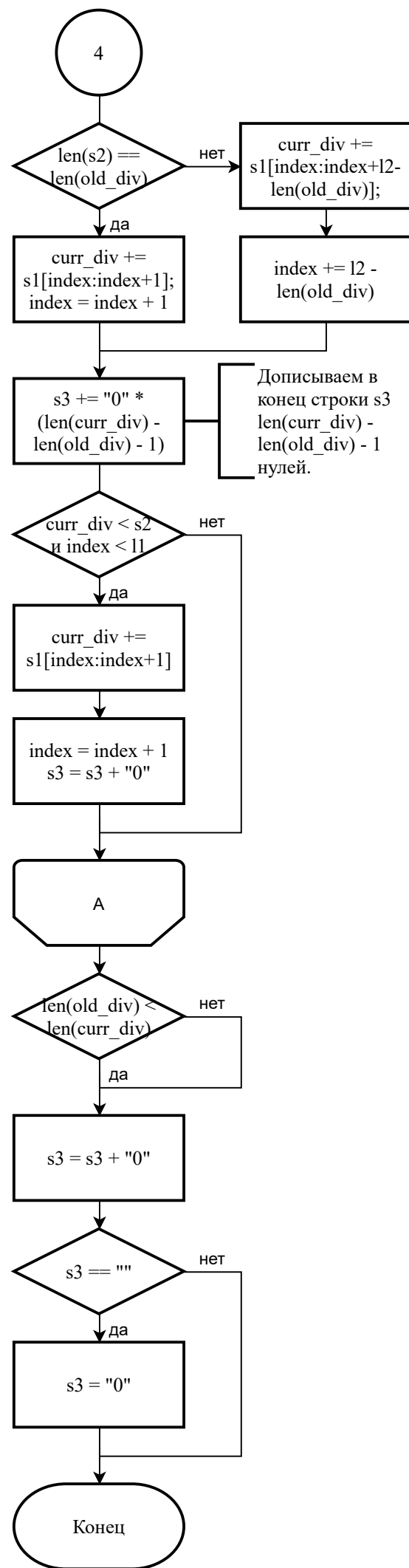
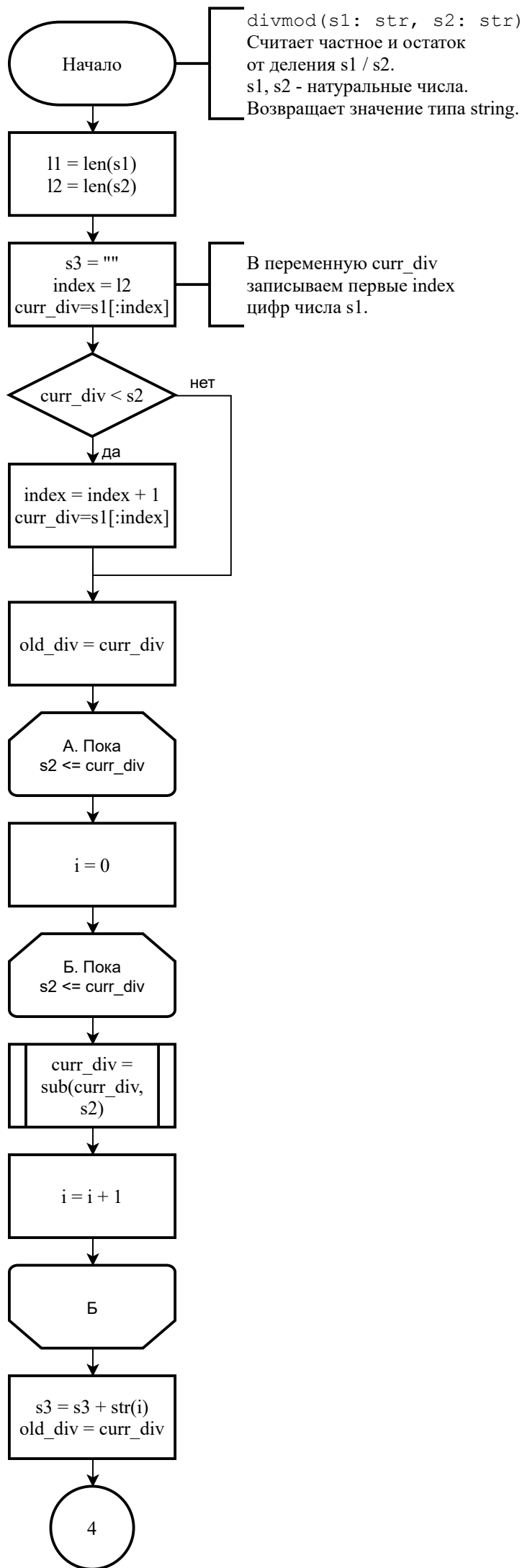
Ниже представлены блок-схемы функций, используемых для работы с длинными числами.

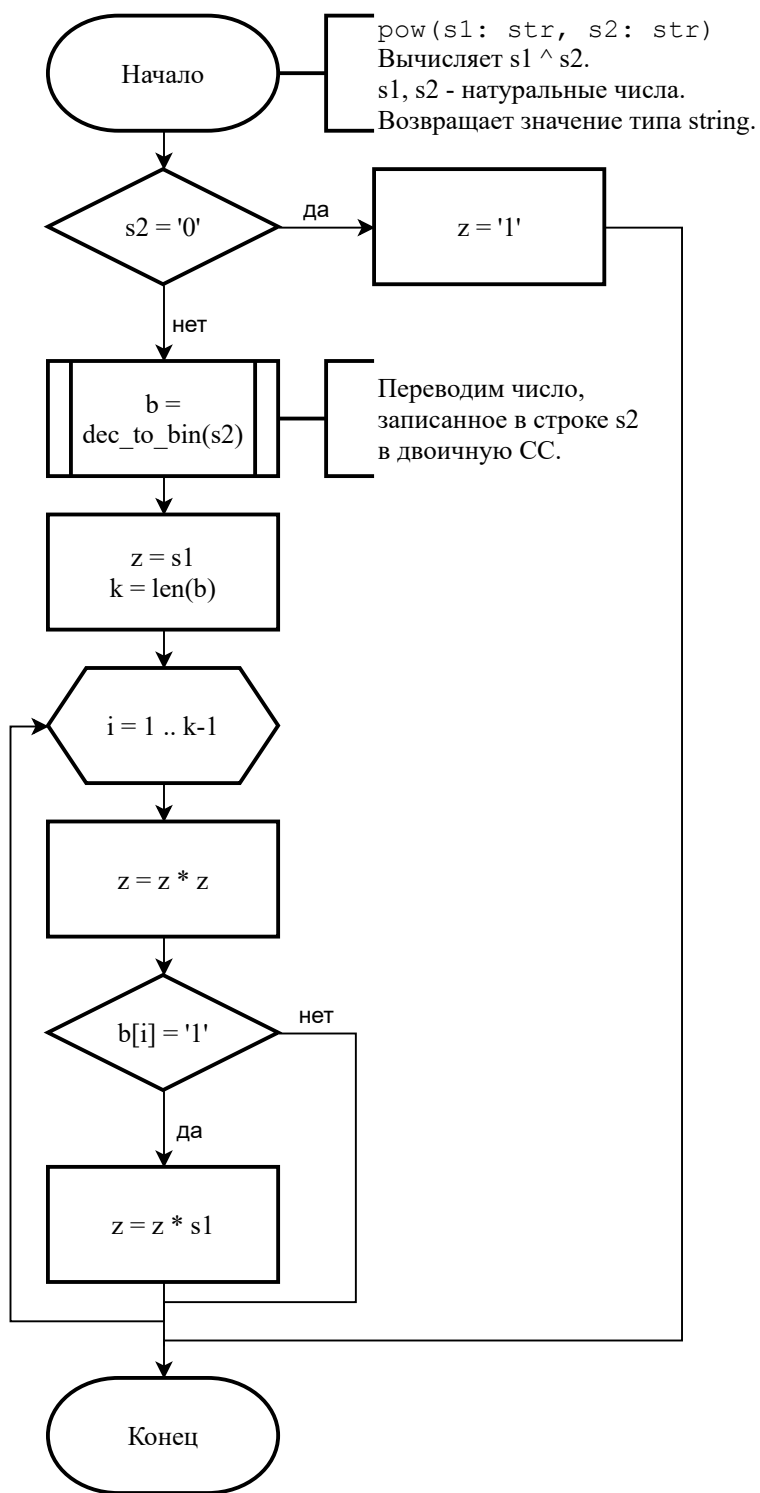


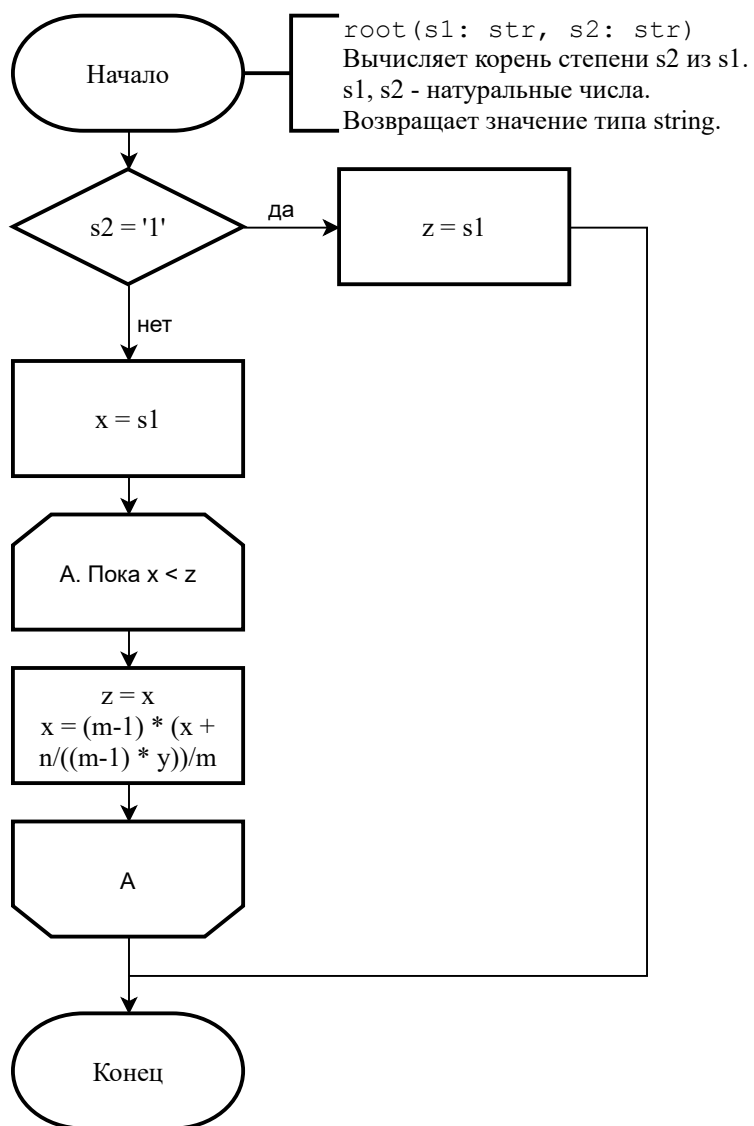


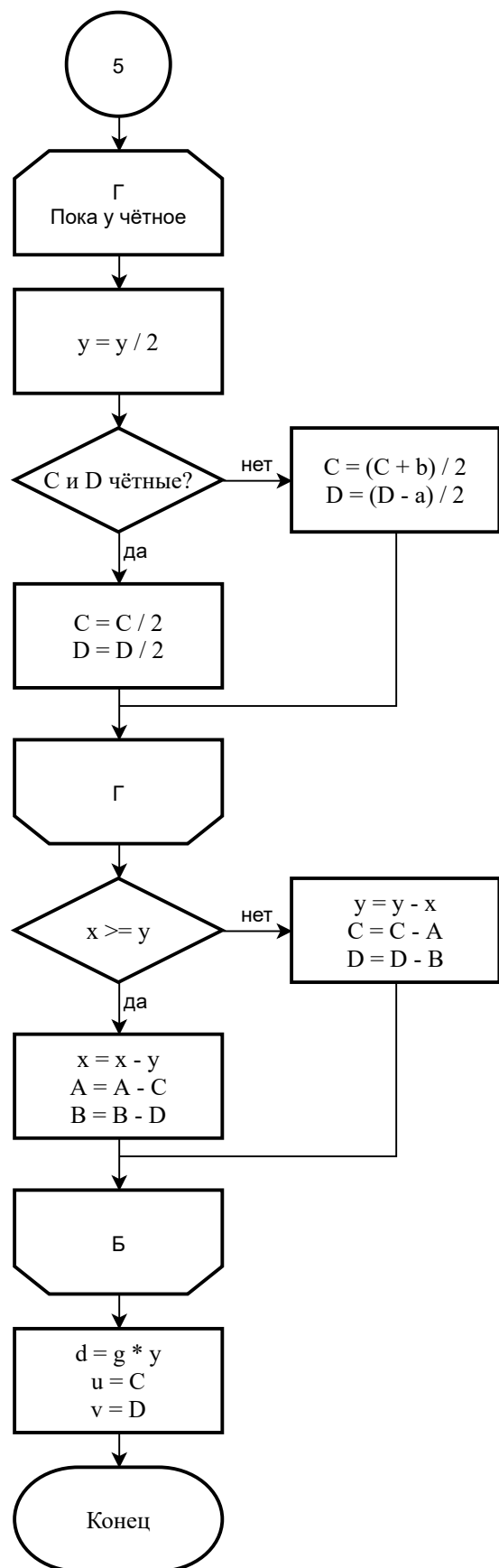
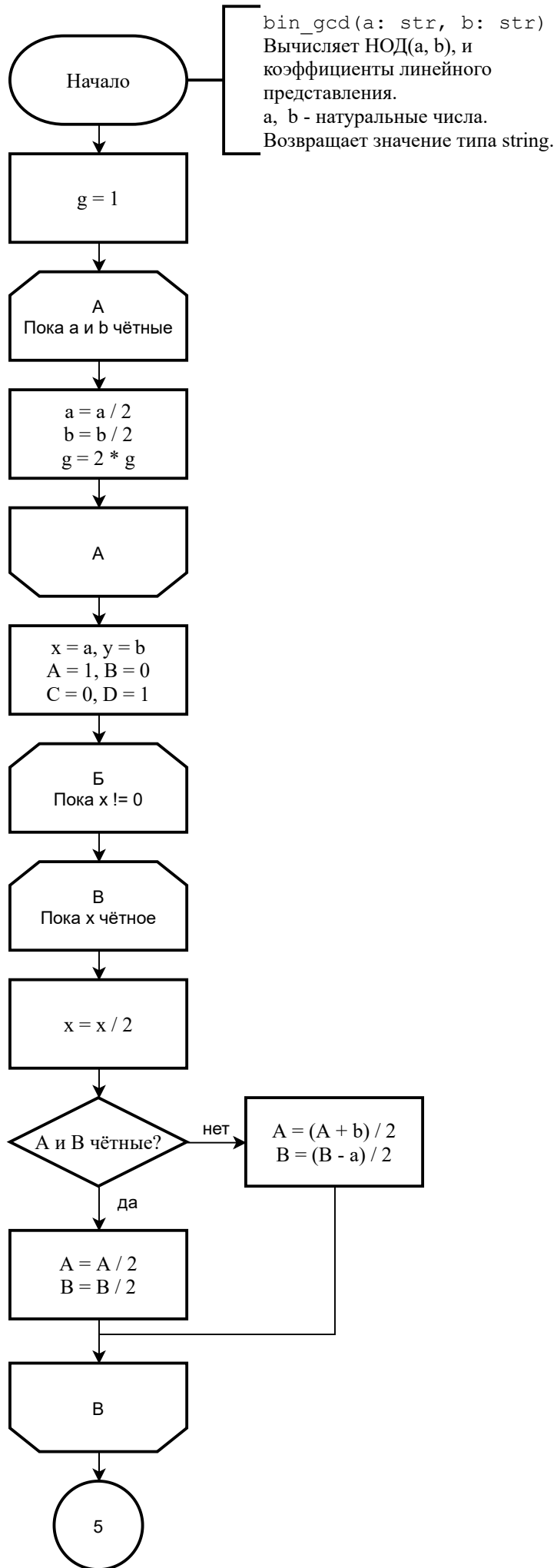


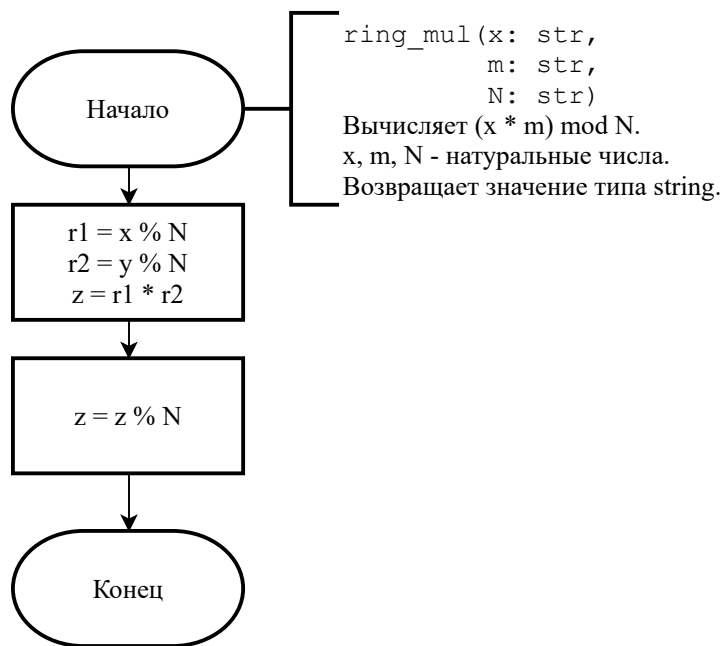
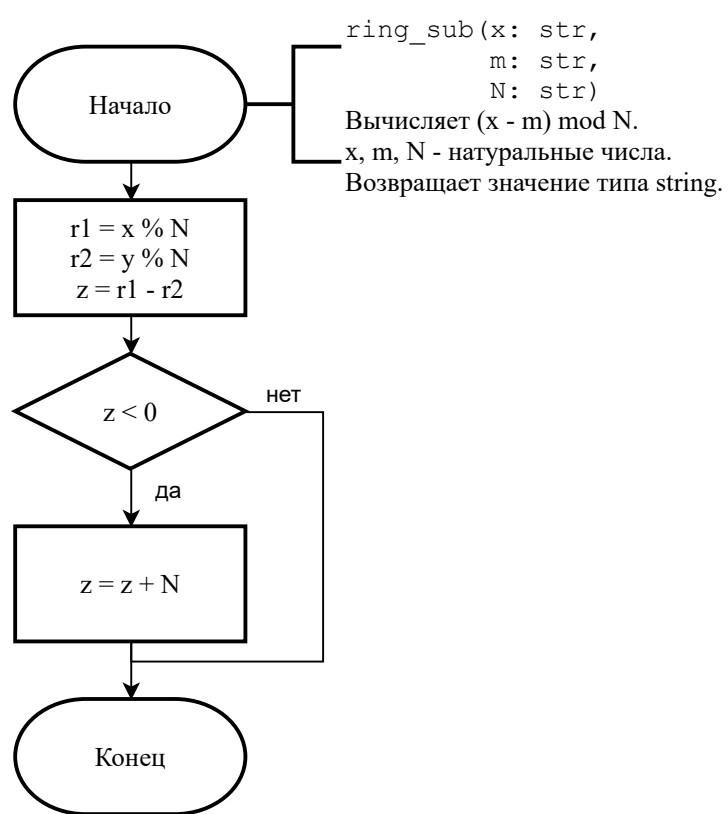
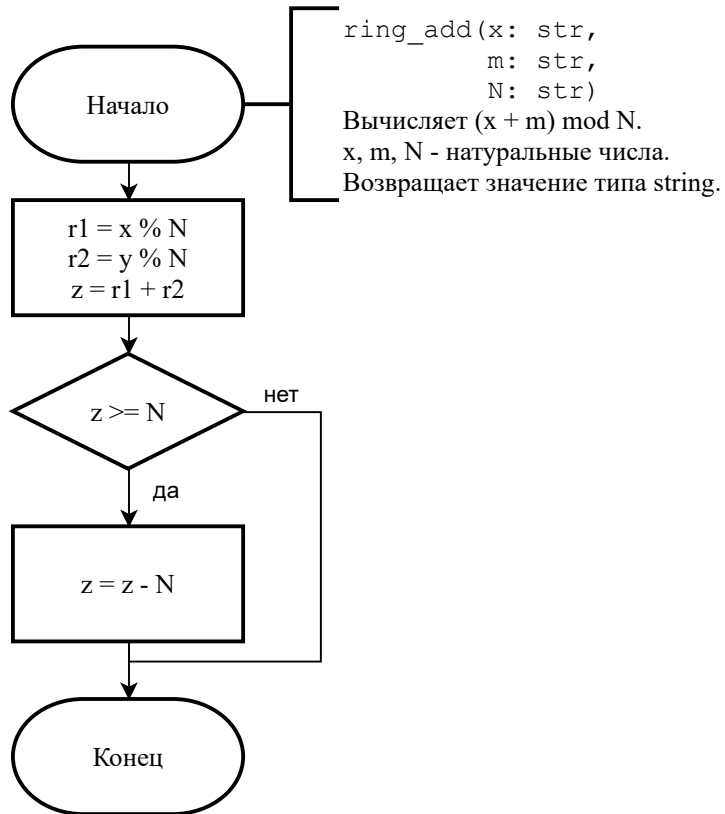


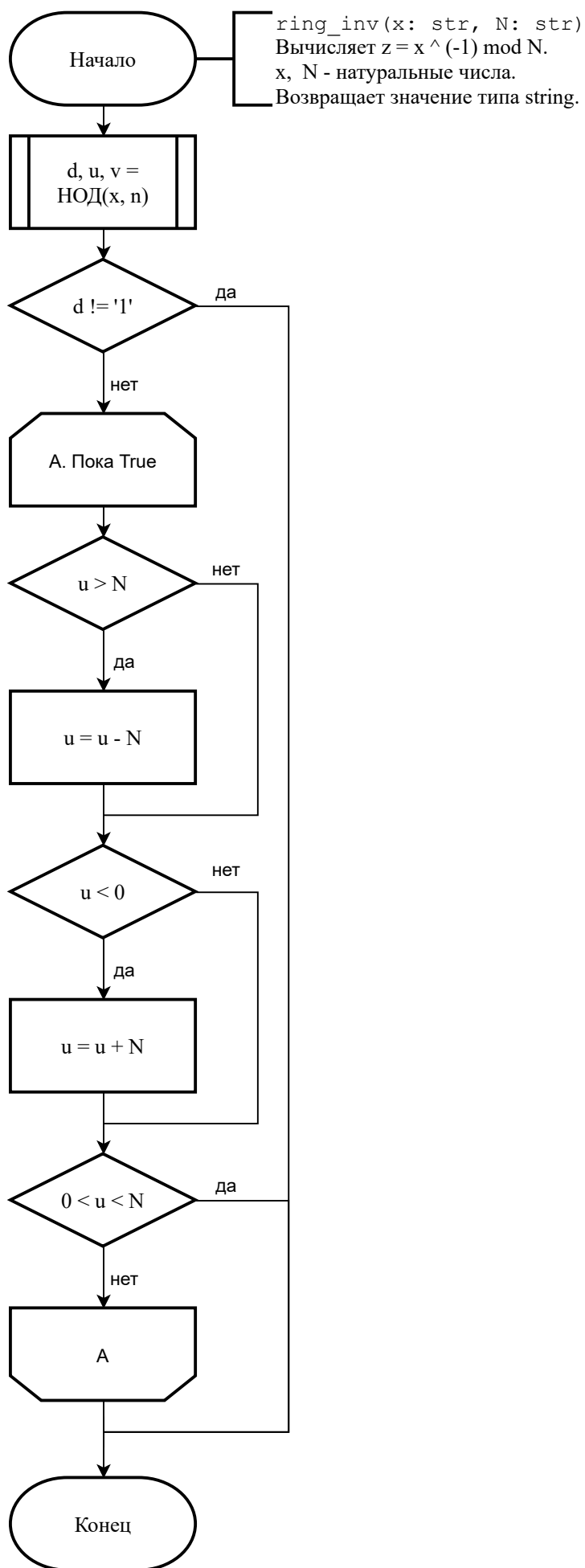


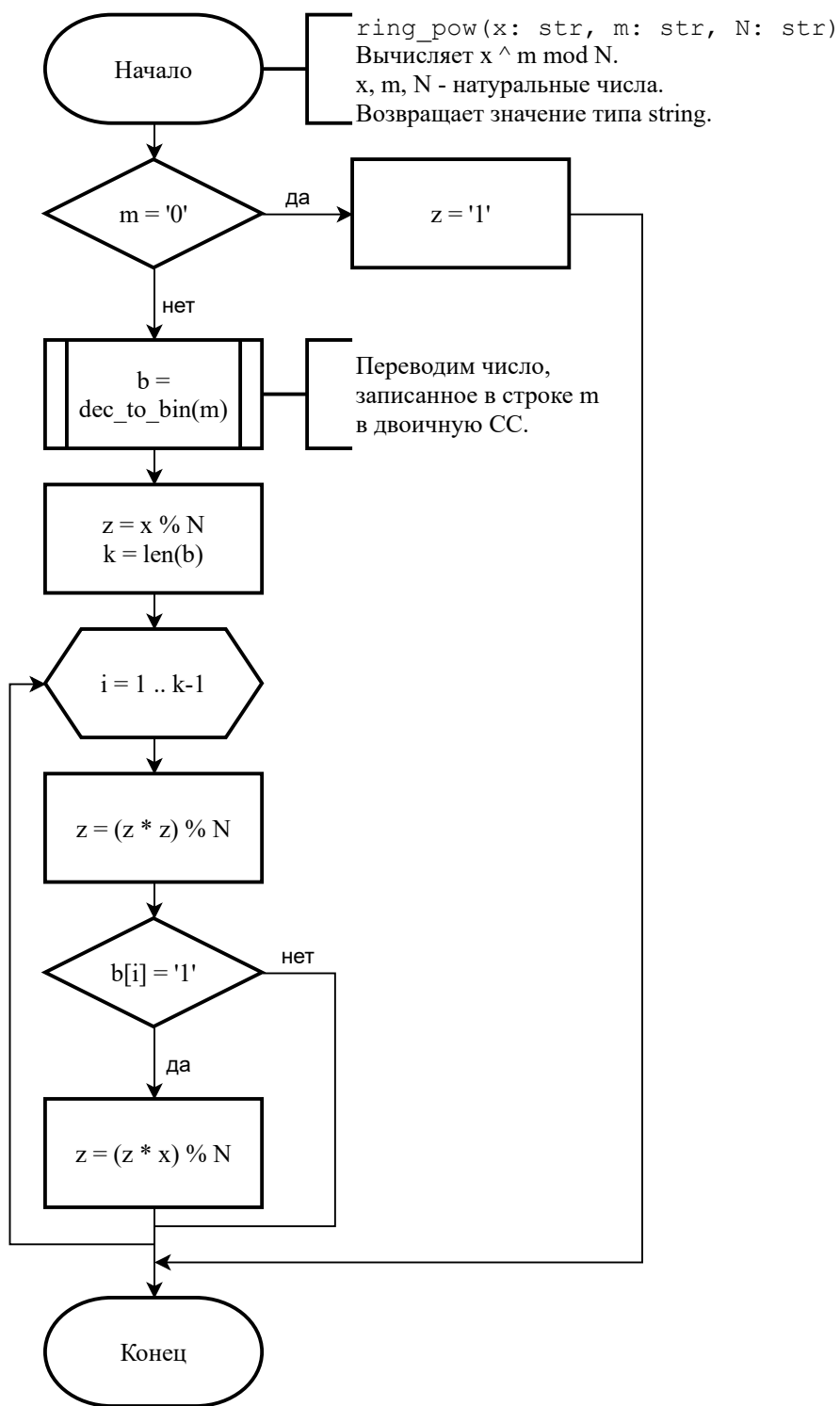












6 Примеры работы программы

Инициализируем несколько переменных объектами типа *BigInt* и посмотрим на результаты арифметических операций:

```
1 from bigint import BigInt
2
3 x1 = BigInt('9999999999999999999')
4 y1 = BigInt('1111111111111111111')
5 x2 = BigInt('99')
6 y2 = BigInt('11')
7 N = BigInt('12345678998765432')
8
9 print('x1 =', x1)
10 print('y1 =', y1)
11 print('x2 =', x2)
12 print('y2 =', y2)
13 print('N =', N)
14
15 print('x1 + y1 =', x1 + y1)
16 print('x1 - y1 =', x1 - y1)
17 print('x1 * y1 =', x1 * y1)
18 print('x1 / y1 =', x1 / y1)
19 print('x1 mod y1 =', x1 % y1)
20 print('x2 ^ y2 =', x2 ** y2)
21 print('Корень из x2 степени y2 = ', BigInt.root(x2, y2))
22 print('НОД(x1, y1) =', BigInt.gcd(x1, y1)[0])
23 print('(x1 + y1) mod N =', BigInt.ring_add(x1, y1, N))
24 print('(x1 - y1) mod N =', BigInt.ring_sub(x1, y1, N))
25 print('(x1 * y1) mod N =', BigInt.ring_mul(x1, y1, N))
26 print('x1^(-1) mod N =', BigInt.ring_inv(x1, N))
27 print('x2^y2 mod N =', BigInt.ring_pow(x2, y2, N))
```

Вывод программы:

- $x1 = 9999999999999999999$
- $y1 = 1111111111111111111$
- $x2 = 99$
- $y2 = 11$
- $N = 12345678998765432$

- $x1 + y1 = 11111111111111111110$
- $x1 - y1 = 88888888888888888888$
- $x1 * y1 = 111111111111111111108888888888888888889$
- $x1 / y1 = 9$
- $x1 \bmod y1 = 0$
- $x2^{y2} = 8953382542587164451099$
- Корень из $x2$ степени $y2 = 1$
- $\text{НОД}(x1, y1) = 11111111111111111111$
- $(x1 + y1) \bmod N = 12222223110$
- $(x1 - y1) \bmod N = 97777778488$
- $(x1 * y1) \bmod N = 21010000081689$
- $x1^{(-1)} \bmod N = 5264157999473642$
- $x2^{y2} \bmod N = 12182065501559763$

7 Список литературы

1. *Завгородний М. Г., Майорова С. П.* Программирование. Криптографические алгоритмы: учебное пособие. — Воронеж : Издательский дом ВГУ, 2018.
2. nth root - Wikipedia. — 2021. — URL: https://en.wikipedia.org/wiki/Nth_root.
3. Python 3.9.2 documentation. — 2021. — URL: <https://docs.python.org/3.9/>.
4. Binary GCD algorithm - Wikipedia. — 2021. — URL: https://en.wikipedia.org/wiki/Binary_GCD_algorithm.
5. Extended Euclidean algorithm - Wikipedia. — 2021. — URL: https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm.

8 Исходный код

Ниже приведён исходный код программы. К сожалению, пакет *listings* для ЛАТ_EX очень плохо работает с русскими символами. Из-за этого русскоязычная часть программы стала плохо читаема.

8.1 bigint.py

```
1 from long_math import (dec_to_bin, is_even, l_add, l_divmod, l_mul, l_pow,
2                        l_root, l_sub)
3
4
5 class BigInt:
6
7     def __init__(self, value='0'):
8         if not isinstance(value, str):
9             t = type(value).__name__
10            raise TypeError(f'BigInt() argument must be a string, not "{t}"')
11
12        if value == '-0':
13            value = '0'
14
15        self.is_neg = value[0] == '-'
16        self.value = value[self.is_neg:]
17
18        if not self.value.isdigit():
19            raise TypeError(f'invalid argument for BigInt(): "{value}"')
20
21    def __abs__(self):
22        return BigInt(self.value)
23
24    def __bool__(self):
25        return self.value != '0'
26
27    def __repr__(self):
28        return self.__str__()
29
30    def __str__(self):
31        return ('-' if self.is_neg else '') + self.value
32
33    def __len__(self):
```

```

34         return len(self.value)
35
36     def __eq__(self, other):
37         if isinstance(other, int):
38             other = BigInt(str(other))
39         return self.value == other.value and self.is_neg == other.is_neg
40
41     def __ne__(self, other):
42         if isinstance(other, int):
43             other = BigInt(str(other))
44         return not self == other
45
46     def __lt__(self, other):
47         if self.is_neg == other.is_neg:
48             self_len = len(self)
49             other_len = len(other)
50             if self_len == other_len:
51                 return (self.value < other.value) ^ self.is_neg
52             return (self_len < other_len) ^ self.is_neg
53         return self.is_neg
54
55     def __le__(self, other):
56         return self < other or self == other
57
58     def __gt__(self, other):
59         return not self <= other
60
61     def __ge__(self, other):
62         return not self < other
63
64     def __pos__(self):
65         return BigInt(('-' if self.is_neg else '') + self.value)
66
67     def __neg__(self):
68         return BigInt(('' if self.is_neg else '-') + self.value)
69
70     def __add__(self, other):
71         if self.is_neg == other.is_neg:
72             result = l_add(self.value, other.value)
73             return BigInt(('-' if self.is_neg else '') + result)
74         x, y = sorted((abs(self), abs(other)))

```

```

75     neg = max((self, other), key=lambda e: abs(e)).is_neg
76     return BigInt(('-' if neg else '') + (y - x).value)
77
78     def __sub__(self, other):
79         if not self.is_neg and not other.is_neg:
80             y, x = sorted((self, other))
81             result = l_sub(x.value, y.value)
82             return BigInt(('-' if self < other else '') + result)
83
84         if self.is_neg and not other.is_neg:
85             return BigInt('-' + (abs(self) + abs(other)).value)
86
87         if not self.is_neg and other.is_neg:
88             return BigInt((abs(self) + abs(other)).value)
89
90         if self.is_neg and other.is_neg:
91             return self + abs(other)
92
93     def __mul__(self, other):
94         result = l_mul(self.value, other.value)
95         return BigInt(('-' if self.is_neg != other.is_neg else '') + result)
96
97     def __truediv__(self, other):
98         if other.value == '0':
99             raise ZeroDivisionError('division by zero')
100         result = l_divmod(self.value, other.value)[0]
101         return BigInt(('' if self.is_neg == other.is_neg else '-') + result)
102
103     def __mod__(self, other):
104         if other.value == '0':
105             raise ZeroDivisionError('division by zero')
106         mod = l_divmod(self.value, other.value)[1]
107         mod = BigInt(mod)
108
109         if mod.value == '0':
110             return mod
111
112         return {
113             not self.is_neg and not other.is_neg: mod,
114             self.is_neg and not other.is_neg: other - mod,
115             not self.is_neg and other.is_neg: mod + other,

```

```

116         self.is_neg and other.is_neg: -mod
117     }[True]
118
119     def __pow__(self, other):
120         result = l_pow(self.value, other.value)
121         if int(other.value[-1]) % 2:
122             return BigInt(('-' if self.is_neg else '')) + result)
123         return BigInt(result)
124
125     @staticmethod
126     def root(a, b):
127         result = l_root(a.value, b.value)
128         return BigInt(result)
129
130     @staticmethod
131     def gcd(a, b):
132         if a.value == '0':
133             return b
134         if b.value == '0':
135             return a
136         a = BigInt(a.value)
137         b = BigInt(b.value)
138         zero, one = BigInt('0'), BigInt('1')
139         r, old_r = a, b
140         s, old_s = zero, one
141         t, old_t = one, zero
142         while r:
143             q = old_r / r
144             old_r, r = r, old_r - q * r
145             old_s, s = s, old_s - q * s
146             old_t, t = t, old_t - q * t
147         return old_r, old_t, old_s
148
149     @staticmethod
150     def bin_gcd(a, b):
151         a = BigInt(a.value)
152         b = BigInt(b.value)
153         zero, one, two = BigInt('0'), BigInt('1'), BigInt('2')
154         g = one
155         while is_even(a.value) and is_even(b.value):
156             a /= two

```

```

157         b /= two
158         g *= two
159     x, y = a, b
160     A, B, C, D = one, zero, zero, one
161     while x:
162         while is_even(x.value):
163             x /= two
164             if is_even(A.value) and is_even(B.value):
165                 A /= two
166                 B /= two
167             else:
168                 A = (A + b) / two
169                 B = (B - a) / two
170         while is_even(y.value):
171             y /= two
172             if is_even(C.value) and is_even(D.value):
173                 C /= two
174                 D /= two
175             else:
176                 C = (C + b) / two
177                 D = (D - a) / two
178         if x < y:
179             y -= x
180             C -= A
181             D -= B
182         else:
183             x -= y
184             A -= C
185             B -= D
186     return g * y, C, D
187
188     @staticmethod
189     def lsbgcd(a, b):
190         a = BigInt(a.value)
191         b = BigInt(b.value)
192         is_swap = False
193         if b > a:
194             a, b = b, a
195             is_swap = True
196         zero, one, two = BigInt('0'), BigInt('1'), BigInt('2')
197         x, y = a, b

```

```

198     A, B, C, D = one, zero, zero, one
199     log2_10 = BigInt('3')
200     n = log2_10 * BigInt(str(len(y)))
201     while y:
202         two_n = two ** n
203         left = two_n * y
204         right = two_n * two * y
205         while True:
206             if left <= x < right:
207                 break
208             if x < left:
209                 n -= one
210             if x >= right:
211                 n += one
212             two_n = two ** n
213             left = two_n * y
214             right = two_n * two * y
215         s = x - left
216         p = right - x
217         if s <= p:
218             t = s
219             At = A - two_n * C
220             Bt = B - two_n * D
221         else:
222             t = p
223             At = two_n * two * C - A
224             Bt = two_n * two * D - B
225         if t <= y:
226             x = y
227             y = t
228             A, B, C, D = C, D, At, Bt
229         else:
230             x = t
231             A = At
232             B = Bt
233     if is_swap:
234         return x, B, A # d, v, u
235     return x, A, B # d, u, v
236
237     @staticmethod
238     def ring_add(x, y, n):

```

```

239         r1 = x % n
240         r2 = y % n
241         z = r1 + r2
242         if z >= n:
243             z -= n
244         return z
245
246     @staticmethod
247     def ring_sub(x, y, n):
248         r1 = x % n
249         r2 = y % n
250         z = r1 - r2
251         if z < BigInt('0'):
252             z += n
253         return z
254
255     @staticmethod
256     def ring_mul(x, y, n):
257         r1 = x % n
258         r2 = y % n
259         z = r1 * r2
260         return z % n
261
262     @staticmethod
263     def ring_inv(x, n):
264         if x.value == '1':
265             return x
266         d, v, u = BigInt.lsbgcd(x, n)
267         if d != BigInt('1'):
268             return None
269         zero = BigInt('0')
270         while True:
271             if u > n:
272                 u -= n
273             if u < zero:
274                 u += n
275             if zero < u < n:
276                 break
277         return u
278
279     @staticmethod

```

```

280     def ring_pow(x, m, n):
281         if m.value == '0':
282             return BigInt('1')
283         b = dec_to_bin(m.value)
284         z = x % n
285         for i in range(1, len(b)):
286             z = (z * z) % n
287             if b[i] == '1':
288                 z = (z * x) % n
289         return z
290
291
292 if __name__ == '__main__':
293     menu_text = '\n'.join([
294         'Выберите операцию:',
295         '1)  $x + y$ ',
296         '2)  $x - y$ ',
297         '3)  $x * y$ ',
298         '4)  $x / y$ ',
299         '5)  $x \bmod y$ ',
300         '6)  $x ^ y$ ',
301         '7) Корень из  $X$  степени  $Y$ ',
302         '8) НОД( $x, y$ )',
303         '9)  $(x + y) \bmod N$ ',
304         '10)  $(x - y) \bmod N$ ',
305         '11)  $(x * y) \bmod N$ ',
306         '12)  $x^{(-1)} \bmod N$ ',
307         '13)  $x^y \bmod N$ ',
308     ])
309     print(menu_text)
310     choice = input('Введите номер операции: ')
311
312     if int(choice) < 1 or int(choice) > 13:
313         print('Выбрано несуществующее значение: (')
314         input('Для выхода нажмите Enter...')
315         exit(0)
316
317     x = BigInt(input('Введите первое число(x): '))
318     if choice != '12':
319         y = BigInt(input('Введите второе число(y): '))
320     if 9 <= int(choice) <= 13:

```



```

321         n = BigInt(input('Введите модуль(N): '))
322
323     if choice == '1':
324         print('x + y =', x + y)
325     elif choice == '2':
326         print('x - y =', x - y)
327     elif choice == '3':
328         print('x * y =', x * y)
329     elif choice == '4':
330         print('x / y =', x / y)
331     elif choice == '5':
332         print('x mod y =', x % y)
333     elif choice == '6':
334         print('x ^ y =', x ** y)
335     elif choice == '7':
336         print('Корень изX степениY = ', BigInt.root(x, y))
337     elif choice == '8':
338         print('НОД(x, y), u, v =', *BigInt.lsbgcd(x, y))
339     elif choice == '9':
340         print('(x + y) mod N =', BigInt.ring_add(x, y, n))
341     elif choice == '10':
342         print('(x - y) mod N =', BigInt.ring_sub(x, y, n))
343     elif choice == '11':
344         print('(x * y) mod N =', BigInt.ring_mul(x, y, n))
345     elif choice == '12':
346         inv = BigInt.ring_inv(x, n)
347         if inv is None:
348             print(f'Обратный элемент числа{x} по модулю{n} не существует!')
349         else:
350             print('x(-1) mod N =', BigInt.ring_inv(x, n))
351     elif choice == '13':
352         print('xy mod N =', BigInt.ring_pow(x, y, n))
353
354     input('Для выхода нажмите Enter...')

```

8.2 tests.py

```
1 import math
2 import unittest
3 from random import randint
4
5 from bigint import BigInt
6 from long_math import (dec_to_bin, l_add, l_divmod, l_mul, l_pow, l_root,
7                        l_sub, less_than)
8
9
10 class TestLongMath(unittest.TestCase):
11
12     MIN = 10 ** 20
13     MAX = 10 ** 30
14     TESTS_COUNT = 10 ** 5
15
16     def test_add(self):
17         for _ in range(self.TESTS_COUNT):
18             x = randint(self.MIN, self.MAX)
19             y = randint(self.MIN, self.MAX)
20             self.assertEqual(str(x + y), l_add(str(x), str(y)))
21
22     def test_sub(self):
23         for _ in range(self.TESTS_COUNT):
24             x = randint(self.MIN, self.MAX)
25             y = randint(self.MIN, self.MAX)
26             x, y = sorted([x, y], reverse=True)
27             self.assertEqual(str(x - y), l_sub(str(x), str(y)))
28
29     def test_mul(self):
30         for _ in range(self.TESTS_COUNT):
31             x = randint(self.MIN, self.MAX)
32             y = randint(self.MIN, self.MAX)
33             self.assertEqual(str(x * y), l_mul(str(x), str(y)))
34
35     def test_divmod(self):
36         for _ in range(self.TESTS_COUNT):
37             x = randint(self.MIN, self.MAX)
38             y = randint(self.MIN, self.MAX)
39             self.assertEqual(tuple(map(str, divmod(x, y))), l_divmod(str(x), str(y)))
40
```

```

41     def test_pow(self):
42         for _ in range(100):
43             x = randint(0, 1000)
44             y = randint(0, 1000)
45             self.assertEqual(str(x ** y), l_pow(str(x), str(y)))
46
47     def test_root(self):
48         for _ in range(100):
49             x = randint(0, 100)
50             y = randint(1, 100)
51             self.assertEqual(str(int(x ** (1 / y))), l_root(str(x), str(y)))
52
53     def test_dec_to_bin(self):
54         for _ in range(self.TESTS_COUNT):
55             x = randint(self.MIN, self.MAX)
56             self.assertEqual(bin(x)[2:], dec_to_bin(x))
57
58     def test_less_than(self):
59         for _ in range(self.TESTS_COUNT):
60             x = randint(self.MIN, self.MAX)
61             y = randint(self.MIN, self.MAX)
62             self.assertEqual(x < y, less_than(str(x), str(y)))
63
64
65 class TestBigInt(unittest.TestCase):
66
67     MIN = -10 ** 30
68     MAX = 10 ** 30
69     TESTS_COUNT = 10 ** 5
70
71     def test_add(self):
72         for _ in range(self.TESTS_COUNT):
73             x = randint(self.MIN, self.MAX)
74             y = randint(self.MIN, self.MAX)
75             big_x = BigInt(str(x))
76             big_y = BigInt(str(y))
77             self.assertEqual(x + y, big_x + big_y)
78
79     def test_sub(self):
80         for _ in range(self.TESTS_COUNT):
81             x = randint(self.MIN, self.MAX)

```

```

82         y = randint(self.MIN, self.MAX)
83         big_x = BigInt(str(x))
84         big_y = BigInt(str(y))
85         self.assertEqual(x - y, big_x - big_y)
86
87     def test_mul(self):
88         for _ in range(self.TESTS_COUNT):
89             x = randint(self.MIN, self.MAX)
90             y = randint(self.MIN, self.MAX)
91             big_x = BigInt(str(x))
92             big_y = BigInt(str(y))
93             self.assertEqual(x * y, big_x * big_y)
94
95     def test_div(self):
96         for _ in range(self.TESTS_COUNT):
97             x = randint(self.MIN, self.MAX)
98             y = randint(self.MIN, self.MAX)
99             big_x = BigInt(str(x))
100            big_y = BigInt(str(y))
101            self.assertEqual(int(x / y), big_x / big_y)
102
103     def test_mod(self):
104         for _ in range(self.TESTS_COUNT):
105             x = randint(self.MIN, self.MAX)
106             y = randint(self.MIN, self.MAX)
107             big_x = BigInt(str(x))
108             big_y = BigInt(str(y))
109             self.assertEqual(x % y, big_x % big_y)
110
111     def test_pow(self):
112         for _ in range(100):
113             x = randint(-100, 100)
114             y = randint(0, 100)
115             big_x = BigInt(str(x))
116             big_y = BigInt(str(y))
117             self.assertEqual(x ** y, big_x ** big_y)
118
119     def test_root(self):
120         for _ in range(100):
121             x = randint(0, 100)
122             y = randint(1, 100)

```

```

123         big_x = BigInt(str(x))
124         big_y = BigInt(str(y))
125         self.assertEqual(int(x ** (1 / y)), BigInt.root(big_x, big_y))
126
127     def test_gcd(self):
128         for _ in range(10000):
129             x = randint(0, 10 ** 20)
130             y = randint(0, 10 ** 20)
131             big_x = BigInt(str(x))
132             big_y = BigInt(str(y))
133             d, big_u, big_v = BigInt.gcd(big_x, big_y)
134             u = int(('-' if big_u.is_neg else '')) + big_u.value
135             v = int(('-' if big_v.is_neg else '')) + big_v.value
136             self.assertEqual(str(math.gcd(x, y)), d.value)
137             self.assertEqual(str(u*x + v*y), d.value)
138
139     def test_lsbgcd(self):
140         for _ in range(1000):
141             x = randint(0, 10 ** 20)
142             y = randint(0, 10 ** 20)
143             big_x = BigInt(str(x))
144             big_y = BigInt(str(y))
145             d, big_u, big_v = BigInt.lsbgcd(big_x, big_y)
146             u = int(('-' if big_u.is_neg else '')) + big_u.value
147             v = int(('-' if big_v.is_neg else '')) + big_v.value
148             self.assertEqual(str(math.gcd(x, y)), d.value)
149             self.assertEqual(str(u*x + v*y), d.value)
150
151     def test_ring_add(self):
152         values = [
153             ('3', '4', '5', '2'),
154             ('12345678', '87654321', '123', '15'),
155             ('12345678', '0', '987', '282'),
156             ('0', '12345678', '987', '282'),
157             ('9999999999', '9999999999', '9999999999', '0')
158         ]
159         for args in values:
160             args = list(map(BigInt, args))
161             self.assertEqual(BigInt.ring_add(*args[:3]), args[3])
162
163     def test_ring_sub(self):

```

```

164     values = [
165         ('3', '4', '5', '4'),
166         ('12345678', '87654321', '123', '75'),
167         ('12345678', '0', '987', '282'),
168         ('0', '12345678', '987', '705'),
169         ('9999999999', '9999999999', '9999999999', '0')
170     ]
171     for args in values:
172         args = list(map(BigInt, args))
173         self.assertEqual(BigInt.ring_sub(*args[:3]), args[3])
174
175     def test_ring_mul(self):
176         values = [
177             ('3', '4', '5', '2'),
178             ('12345678', '87654321', '123', '3'),
179             ('12345678', '0', '987', '0'),
180             ('0', '12345678', '987', '0'),
181             ('9999999999', '9999999999', '9999999999', '0')
182         ]
183         for args in values:
184             args = list(map(BigInt, args))
185             self.assertEqual(BigInt.ring_mul(*args[:3]), args[3])
186
187     def test_ring_inv(self):
188         values = [
189             ('873372847093', str(10 ** 12), '94559444997'),
190             ('3', '6', None),
191         ]
192         for *args, x in values:
193             args = list(map(BigInt, args))
194             if isinstance(x, str):
195                 x = BigInt(x)
196             self.assertEqual(BigInt.ring_inv(*args[:2]), x)
197
198     def test_ring_pow(self):
199         values = [
200             ('3', '4', '5', '1'),
201             ('18', '50', '873372847093', '194798095869'),
202             ('12345678', '0', '987', '1'),
203             ('0', '12345678', '987', '0'),
204             ('999', '999', '999', '0')

```

```
205         ]
206         for args in values:
207             args = list(map(BigInt, args))
208             self.assertEqual(BigInt.ring_pow(*args[:3]), args[3])
209
210
211 if __name__ == '__main__':
212     unittest.main()
```