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
подпись

Обучающаяся _____ С.Д. Бабошин
подпись

Воронеж 2021

Содержание

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 стпень элементы кольца вычетов.

2 Используемые технологии

Программа написана на языке программирования Python 3.9. Плюсом 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 содеражатся 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 началов 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) Дописывем 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 вычисление корня используется алгоритм описанный в [1] и [2].

4.7 НОД

Поиск НОД реализован в методе *BigInt.gcd*. Для вычисления НОД используется бинарный алгоритм ([1], [3]).

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

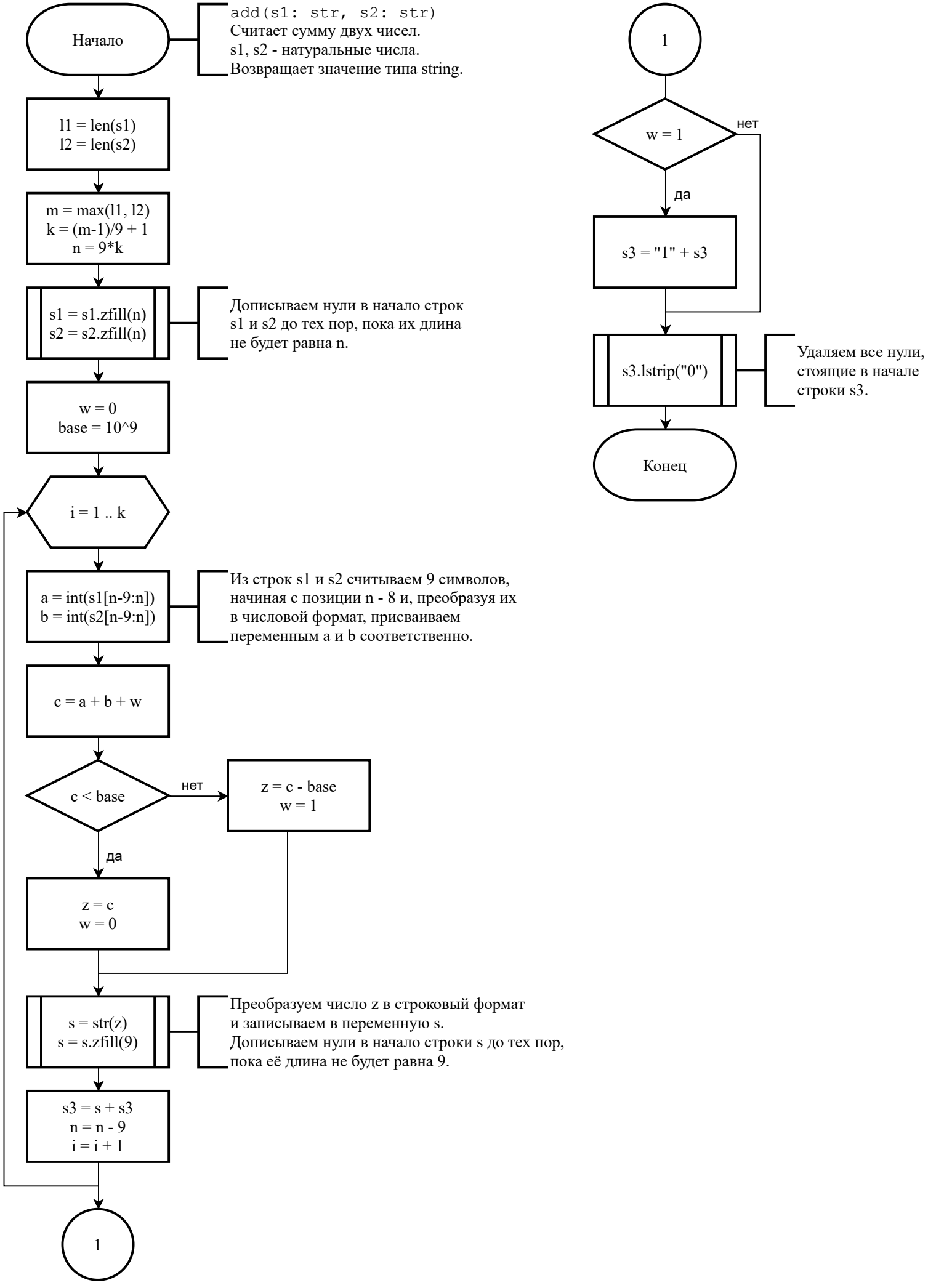
Вычисление обратных элементов реализовано в методе *BigInt.ring_inv*. Для вычисления обратных элементов используется алгоритм описанный в [1].

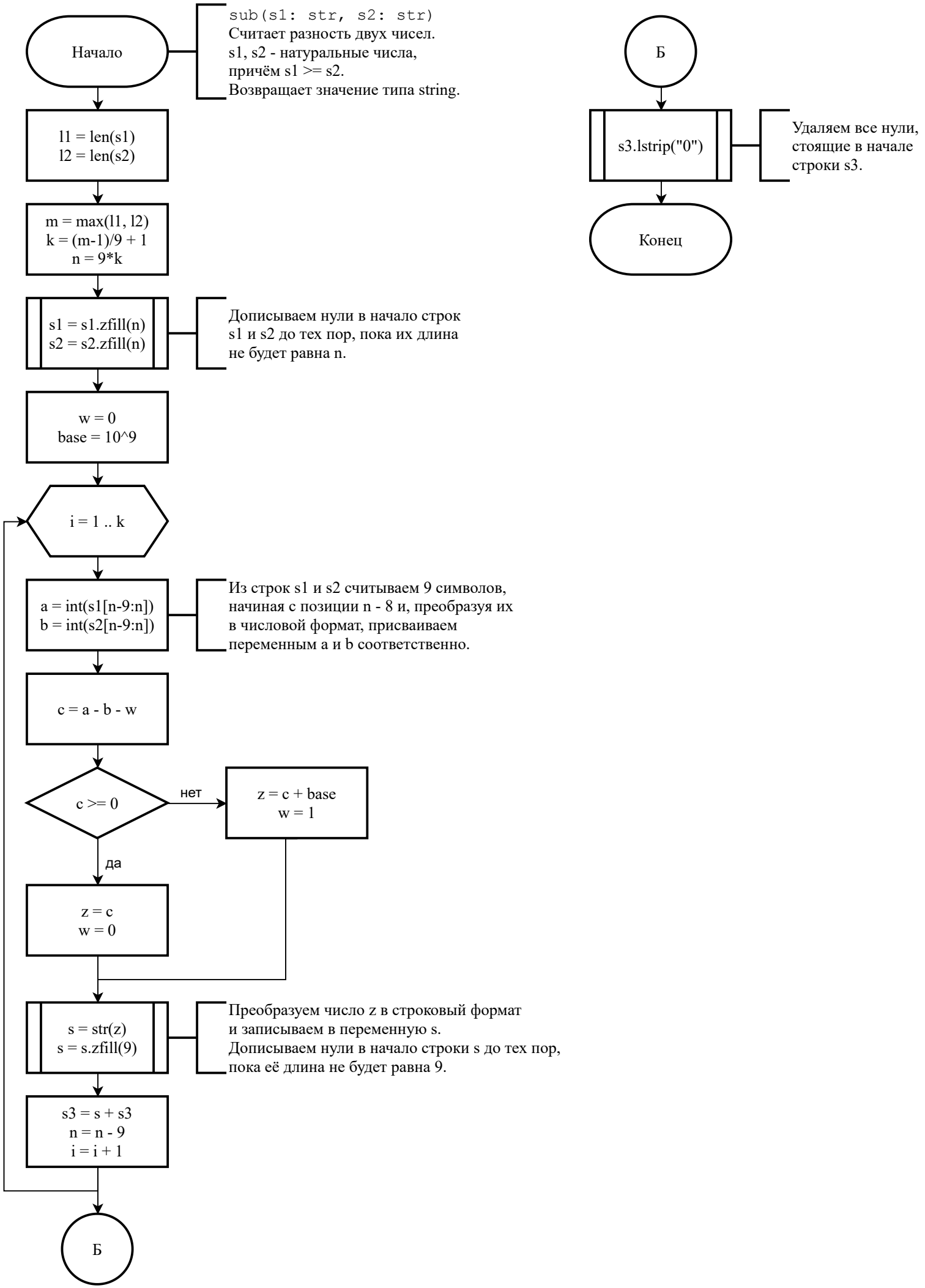
4.12 Возведение в степень в кольце вычетов

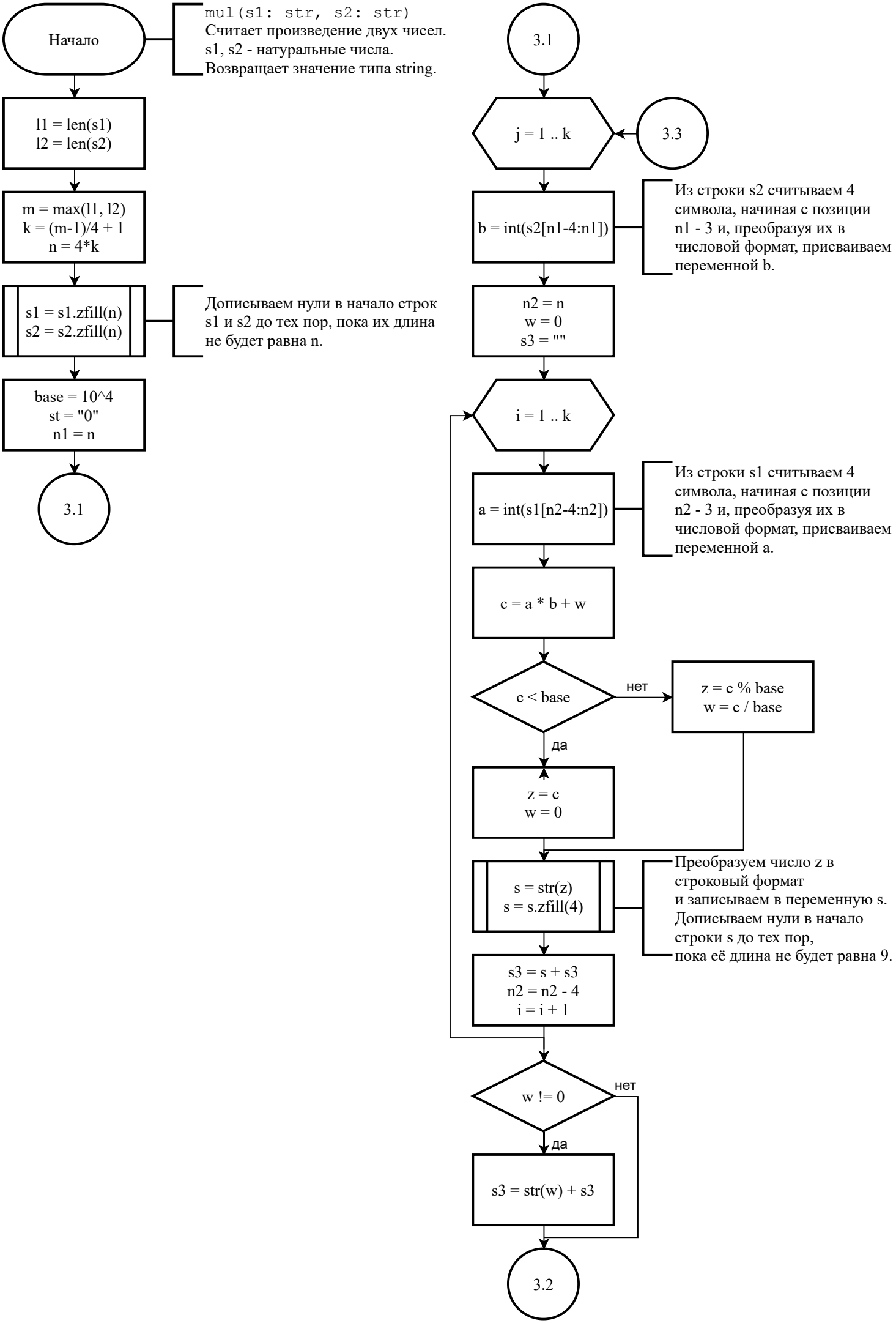
Возведение в степень реализовано в методе *BigInt.ring_pow*. Для возведения в степень используется алгоритм описанный в [1].

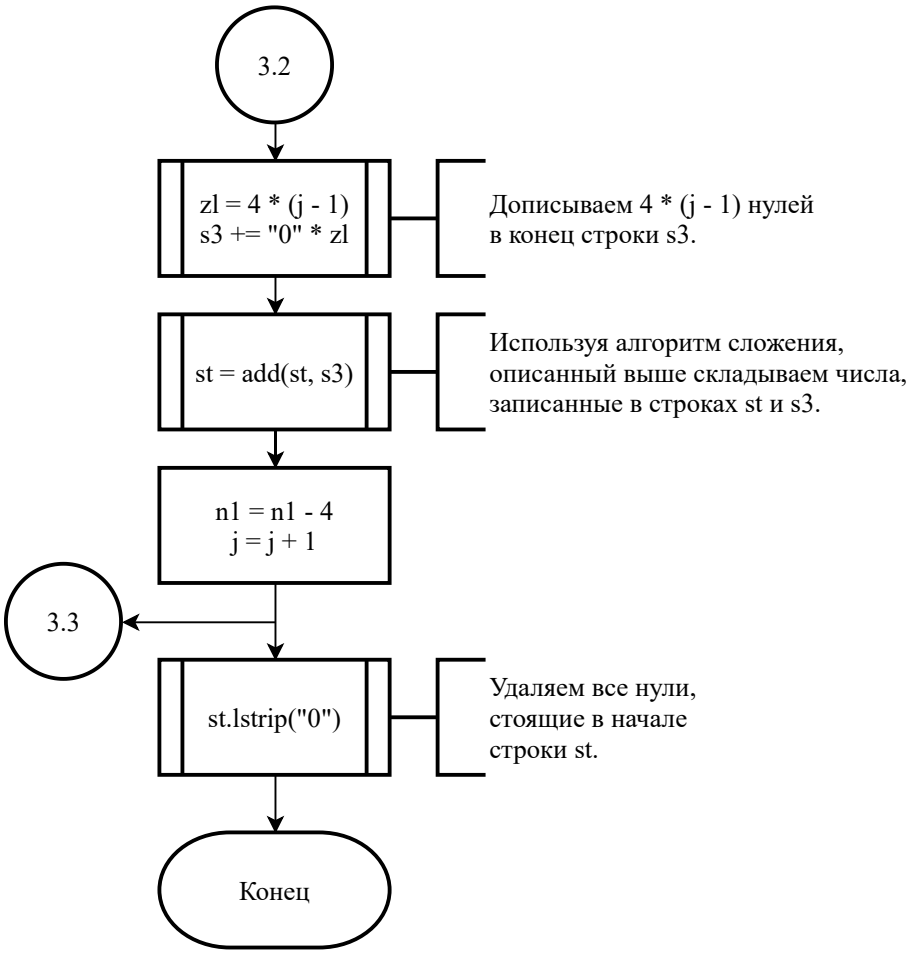
5 Блок-схемы алгоритмов

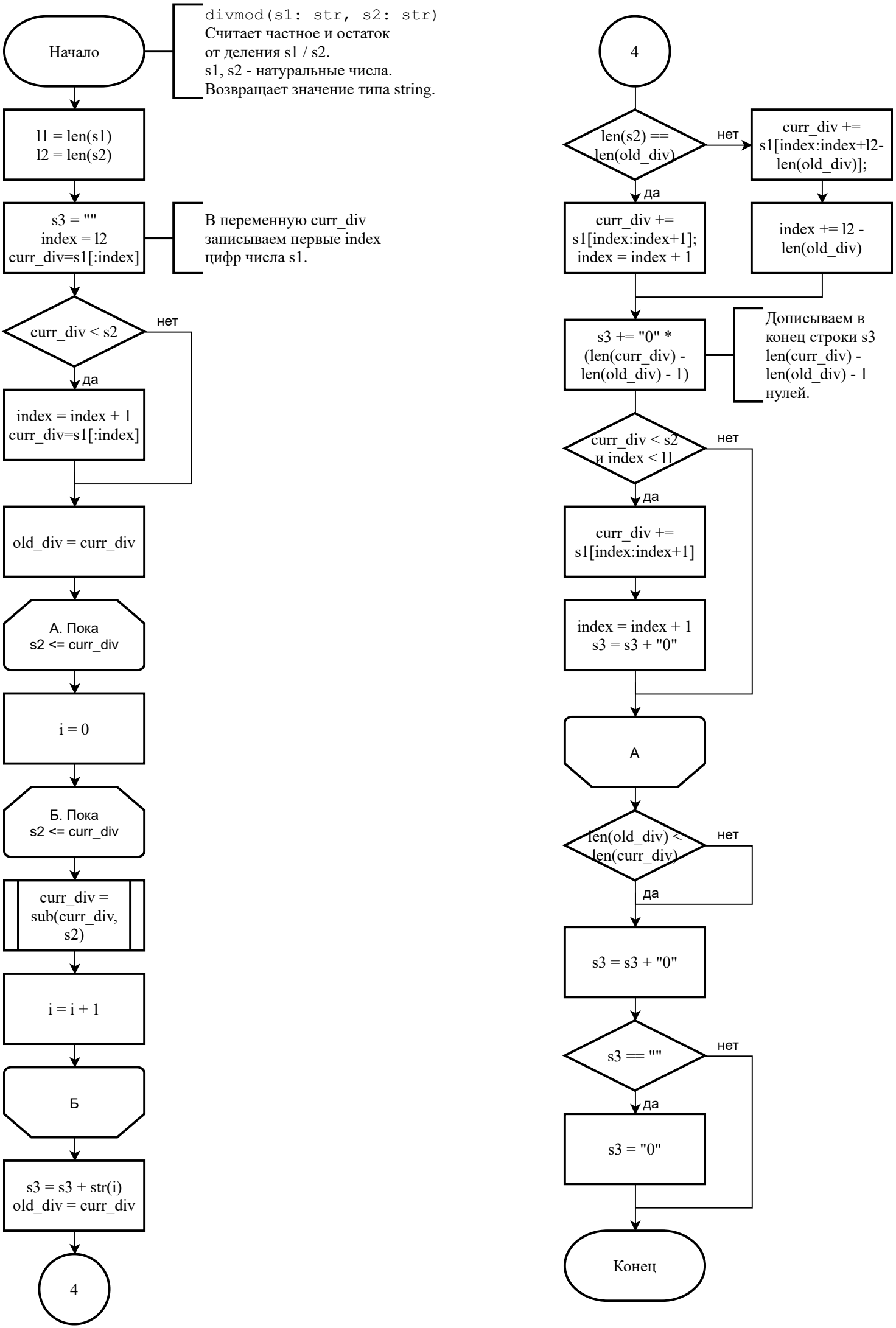
Ниже представлены блок-схемы функций, используемых для работы с длинными числами.


Начало
add(s1: str, s2: str)
Считает сумму двух чисел.
s1, s2 - натуральные числа.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/9 + 1
n = 9*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк s1 и s2 до тех пор, пока их длина не будет равна n.
w = 0
base = 10^9
i = 1 .. k
a = int(s1[n-9:n])
b = int(s2[n-9:n])
Из строк s1 и s2 считываем 9 символов, начиная с позиции n - 8 и, преобразуя их в числовой формат, присваиваем переменным a и b соответственно.
c = a + b + w
c < base
нет
z = c - base
w = 1
да
z = c
w = 0
s = str(z)
s = s.zfill(9)
Преобразуем число z в строковый формат и записываем в переменную s.
Дописываем нули в начало строки s до тех пор, пока её длина не будет равна 9.
s3 = s + s3
n = n - 9
i = i + 1
1
1
w = 1
нет
да
s3 = "1" + s3
s3.lstrip("0")
Удаляем все нули, стоящие в начале строки s3.
Конец


Начало
sub(s1: str, s2: str)
Считает разность двух чисел.
s1, s2 - натуральные числа,
причём s1 >= s2.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/9 + 1
n = 9*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк
s1 и s2 до тех пор, пока их длина
не будет равна n.
w = 0
base = 10^9
i = 1 .. k
a = int(s1[n-9:n])
b = int(s2[n-9:n])
Из строк s1 и s2 считываем 9 символов,
начиная с позиции n - 8 и, преобразуя их
в числовой формат, присваиваем
переменным a и b соответственно.
c = a - b - w
c >= 0
нет
z = c + base
w = 1
да
z = c
w = 0
s = str(z)
s = s.zfill(9)
Преобразуем число z в строковый формат
и записываем в переменную s.
Дописываем нули в начало строки s до тех пор,
пока её длина не будет равна 9.
s3 = s + s3
n = n - 9
i = i + 1
Б
Б
s3.lstrip("0")
Удаляем все нули,
стоящие в начале
строки s3.
Конец


Начало
mul(s1: str, s2: str)
Считает произведение двух чисел.
s1, s2 - натуральные числа.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/4 + 1
n = 4*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк s1 и s2 до тех пор, пока их длина не будет равна n.
base = 10^4
st = "0"
n1 = n
3.1
3.1
j = 1 .. k
3.3
b = int(s2[n1-4:n1])
Из строки s2 считываем 4 символа, начиная с позиции n1 - 3 и, преобразуя их в числовой формат, присваиваем переменной b.
n2 = n
w = 0
s3 = ""
i = 1 .. k
a = int(s1[n2-4:n2])
Из строки s1 считываем 4 символа, начиная с позиции n2 - 3 и, преобразуя их в числовой формат, присваиваем переменной a.
c = a * b + w
c < base
нет
z = c % base
w = c / base
да
z = c
w = 0
s = str(z)
s = s.zfill(4)
Преобразуем число z в строковый формат и записываем в переменную s. Дописываем нули в начало строки s до тех пор, пока её длина не будет равна 9.
s3 = s + s3
n2 = n2 - 4
i = i + 1
w != 0
нет
да
s3 = str(w) + s3
3.2


3.2
zl = 4 * (j - 1)
s3 += "0" * zl
Дописываем 4 * (j - 1) нулей
в конец строки s3.
st = add(st, s3)
Используя алгоритм сложения,
описанный выше складываем числа,
записанные в строках st и s3.
n1 = n1 - 4
j = j + 1
3.3
st.lstrip("0")
Удаляем все нули,
стоящие в начале
строки st.
Конец


Начало
divmod(s1: str, s2: str)
Считает частное и остаток
от деления s1 / s2.
s1, s2 - натуральные числа.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
s3 = ""
index = l2
curr_div=s1[:index]
В переменную curr_div
записываем первые index
цифр числа s1.
curr_div < s2
нет
да
index = index + 1
curr_div=s1[:index]
old_div = curr_div
А. Пока
s2 <= curr_div
i = 0
Б. Пока
s2 <= curr_div
curr_div =
sub(curr_div,
s2)
i = i + 1
Б
s3 = s3 + str(i)
old_div = curr_div
4
4
len(s2) ==
len(old_div)
нет
да
curr_div +=
s1[index:index+l2-
len(old_div)];
index += l2 -
len(old_div)
curr_div +=
s1[index:index+1];
index = index + 1
s3 += "0" *
(len(curr_div) -
len(old_div) - 1)
Дописываем в
конец строки s3
len(curr_div) -
len(old_div) - 1
нулей.
curr_div < s2
и index < l1
нет
да
curr_div +=
s1[index:index+1]
index = index + 1
s3 = s3 + "0"
А
len(old_div) <
len(curr_div)
нет
да
s3 = s3 + "0"
s3 == ""
нет
да
s3 = "0"
Конец

6 Примеры работы программы

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

```
from bigint import BigInt

x1 = BigInt('99999999999999999999')
y1 = BigInt('11111111111111111111')
x2 = BigInt('99')
y2 = BigInt('11')
N = BigInt('12345678998765432')

print('x1 =', x1)
print('y1 =', y1)
print('x2 =', x2)
print('y2 =', y2)
print('N =', N)

print('x1 + y1 =', x1 + y1)
print('x1 - y1 =', x1 - y1)
print('x1 * y1 =', x1 * y1)
print('x1 / y1 =', x1 / y1)
print('x1 mod y1 =', x1 % y1)
print('x2 ^ y2 =', x2 ** y2)
print('Корень изx2 степениу2 = ', BigInt.root(x2, y2))
print('НОД(x1, y1) =', BigInt.gcd(x1, y1)[0])
print('(x1 + y1) mod N =', BigInt.ring_add(x1, y1, N))
print('(x1 - y1) mod N =', BigInt.ring_sub(x1, y1, N))
print('(x1 * y1) mod N =', BigInt.ring_mul(x1, y1, N))
print('x1^(-1) mod N =', BigInt.ring_inv(x1, N))
print('x2^y2 mod N =', BigInt.ring_pow(x2, y2, N))
```

Вывод программы:

- x1 = 99999999999999999999
- y1 = 11111111111111111111
- x2 = 99
- y2 = 11
- N = 12345678998765432

- x1 + y1 = 111111111111111111110
- x1 - y1 = 88888888888888888
- x1 * y1 = 11111111111111111110888888888888888888889
- x1 / y1 = 9
- x1 mod y1 = 0
- $x2^{y2}$ = 8953382542587164451099
- Корень из x2 степени y2 = 1
- НОД(x1, y1) = 11111111111111111111
- (x1 + y1) mod N = 122222223110
- (x1 - y1) mod N = 97777778488
- (x1 * y1) mod N = 21010000081689
- $x1^{(-1)} mod N$ = 5264157999473642
- $x2^{y2} mod N$ = 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. Binary GCD algorithm - Wikipedia. — 2021. — URL: `https://en.wikipedia.org/wiki/Binary_GCD_algorithm`.
4. Python 3.9.2 documentation. — 2021. — URL: `https://docs.python.org/3.9/`.

8 Исходный код

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

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

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

```
116         }[True]
117
118     def __pow__(self, other):
119         result = l_pow(self.value, other.value)
120         if int(other.value[-1]) % 2:
121             return BigInt(('-' if self.is_neg else '') + result)
122         return BigInt(result)
123
124     @staticmethod
125     def root(a, b):
126         result = l_root(a.value, b.value)
127         return BigInt(result)
128
129     @staticmethod
130     def gcd(a, b):
131         if a.value == '0':
132             return b
133         if b.value == '0':
134             return a
135         a = BigInt(a.value)
136         b = BigInt(b.value)
137         zero, one = BigInt('0'), BigInt('1')
138         r, old_r = a, b
139         s, old_s = zero, one
140         t, old_t = one, zero
141         while r:
142             q = old_r / r
143             old_r, r = r, old_r - q * r
144             old_s, s = s, old_s - q * s
145             old_t, t = t, old_t - q * t
146         return old_r, old_t, old_s
147
148     @staticmethod
149     def ring_add(x, y, n):
150         r1 = x % n
151         r2 = y % n
152         z = r1 + r2
153         if z >= n:
154             z -= n
155         return z
156
```

```
157     @staticmethod
158     def ring_sub(x, y, n):
159         r1 = x % n
160         r2 = y % n
161         z = r1 - r2
162         if z < BigInt('0'):
163             z += n
164         return z
165 
166     @staticmethod
167     def ring_mul(x, y, n):
168         r1 = x % n
169         r2 = y % n
170         z = r1 * r2
171         return z % n
172 
173     @staticmethod
174     def ring_inv(x, n):
175         d, v, u = BigInt.gcd(x, n)
176         if d != BigInt('1'):
177             return None
178         zero = BigInt('0')
179         while True:
180             if u > n:
181                 u -= n
182             if u < zero:
183                 u += n
184             if zero < u < n:
185                 break
186         return u
187 
188     @staticmethod
189     def ring_pow(x, m, n):
190         if m.value == '0':
191             return BigInt('1')
192         b = dec_to_bin(m.value)
193         z = x % n
194         for i in range(1, len(b)):
195             z = (z * z) % n
196             if b[i] == '1':
197                 z = (z * x) % n
```

```
198         return z
199
200
201 if __name__ == '__main__':
202     menu_text = '\n'.join([
203         'Выберите операцию:',
204         '1) x + y',
205         '2) x - y',
206         '3) x * y',
207         '4) x / y',
208         '5) x mod y',
209         '6) x ^ y',
210         '7) КореньизX степениY',
211         '8) НОД(x, y)',
212         '9) (x + y) mod N',
213         '10) (x - y) mod N',
214         '11) (x * y) mod N',
215         '12) x^(-1) mod N',
216         '13) x^y mod N',
217     ])
218     print(menu_text)
219     choice = input('Введите номероперации: ')
220
221     if int(choice) < 1 or int(choice) > 13:
222         print('Выбрано несуществующеезначение:(')
223
224     x = BigInt(input('Введите первоечисло(x): '))
225     if choice != '12':
226         y = BigInt(input('Введите второечисло(y): '))
227     if 9 <= int(choice) <= 13:
228         n = BigInt(input('Введите модуль(N): '))
229
230     if choice == '1':
231         print('x + y =', x + y)
232     elif choice == '2':
233         print('x - y =', x - y)
234     elif choice == '3':
235         print('x * y =', x * y)
236     elif choice == '4':
237         print('x / y =', x / y)
238     elif choice == '5':
```

```
239         print('x mod y =', x % y)
240     elif choice == '6':
241         print('x ^ y =', x ** y)
242     elif choice == '7':
243         print('Корень изX степениY = ', BigInt.root(x, y))
244     elif choice == '8':
245         print('НОД(x, y) =', BigInt.gcd(x, y))
246     elif choice == '9':
247         print('(x + y) mod N =', BigInt.ring_add(x, y, n))
248     elif choice == '10':
249         print('(x - y) mod N =', BigInt.ring_sub(x, y, n))
250     elif choice == '11':
251         print('(x * y) mod N =', BigInt.ring_mul(x, y, n))
252     elif choice == '12':
253         inv = BigInt.ring_inv(x, n)
254         if inv is None:
255             print(f'Обратный элементчисла{x} помодулю{n} несуществует!')
256         else:
257             print('x^(-1) mod N =', BigInt.ring_inv(x, n))
258     elif choice == '13':
259         print('x^y mod N =', BigInt.ring_pow(x, y, n))
260 
261     input('Для выходанажмитеEnter...')
```

8.2 tests.py

```
import math
import unittest
from random import randint

from bigint import BigInt
from long_math import (dec_to_bin, l_add, l_divmod, l_mul, l_pow, l_root,
                       l_sub, less_than)


class TestLongMath(unittest.TestCase):

    MIN = 10 ** 20
    MAX = 10 ** 30
    TESTS_COUNT = 10 ** 5

    def test_add(self):
        for _ in range(self.TESTS_COUNT):
            x = randint(self.MIN, self.MAX)
            y = randint(self.MIN, self.MAX)
            self.assertEqual(str(x + y), l_add(str(x), str(y)))

    def test_sub(self):
        for _ in range(self.TESTS_COUNT):
            x = randint(self.MIN, self.MAX)
            y = randint(self.MIN, self.MAX)
            x, y = sorted([x, y], reverse=True)
            self.assertEqual(str(x - y), l_sub(str(x), str(y)))

    def test_mul(self):
        for _ in range(self.TESTS_COUNT):
            x = randint(self.MIN, self.MAX)
            y = randint(self.MIN, self.MAX)
            self.assertEqual(str(x * y), l_mul(str(x), str(y)))

    def test_divmod(self):
        for _ in range(self.TESTS_COUNT):
            x = randint(self.MIN, self.MAX)
            y = randint(self.MIN, self.MAX)
            self.assertEqual(tuple(map(str, divmod(x, y))), l_divmod(str(x), str(y)))

```

```
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_ring_add(self):
140         values = [
141             ('3', '4', '5', '2'),
142             ('12345678', '87654321', '123', '15'),
143             ('12345678', '0', '987', '282'),
144             ('0', '12345678', '987', '282'),
145             ('9999999999', '9999999999', '9999999999', '0')
146         ]
147         for args in values:
148             args = list(map(BigInt, args))
149             self.assertEqual(BigInt.ring_add(*args[:3]), args[3])
150
151     def test_ring_sub(self):
152         values = [
153             ('3', '4', '5', '4'),
154             ('12345678', '87654321', '123', '75'),
155             ('12345678', '0', '987', '282'),
156             ('0', '12345678', '987', '705'),
157             ('9999999999', '9999999999', '9999999999', '0')
158         ]
159         for args in values:
160             args = list(map(BigInt, args))
161             self.assertEqual(BigInt.ring_sub(*args[:3]), args[3])
162
163     def test_ring_mul(self):
```

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