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].

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*). В предложенной реализации содержится ошибка, которую мне не удалось исправить, т. к. она возникает не во всех случаях (примерно 5-10 раз на 100 запусков). Из-за этой ошибки в данном алгоритме в некоторых случаях 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

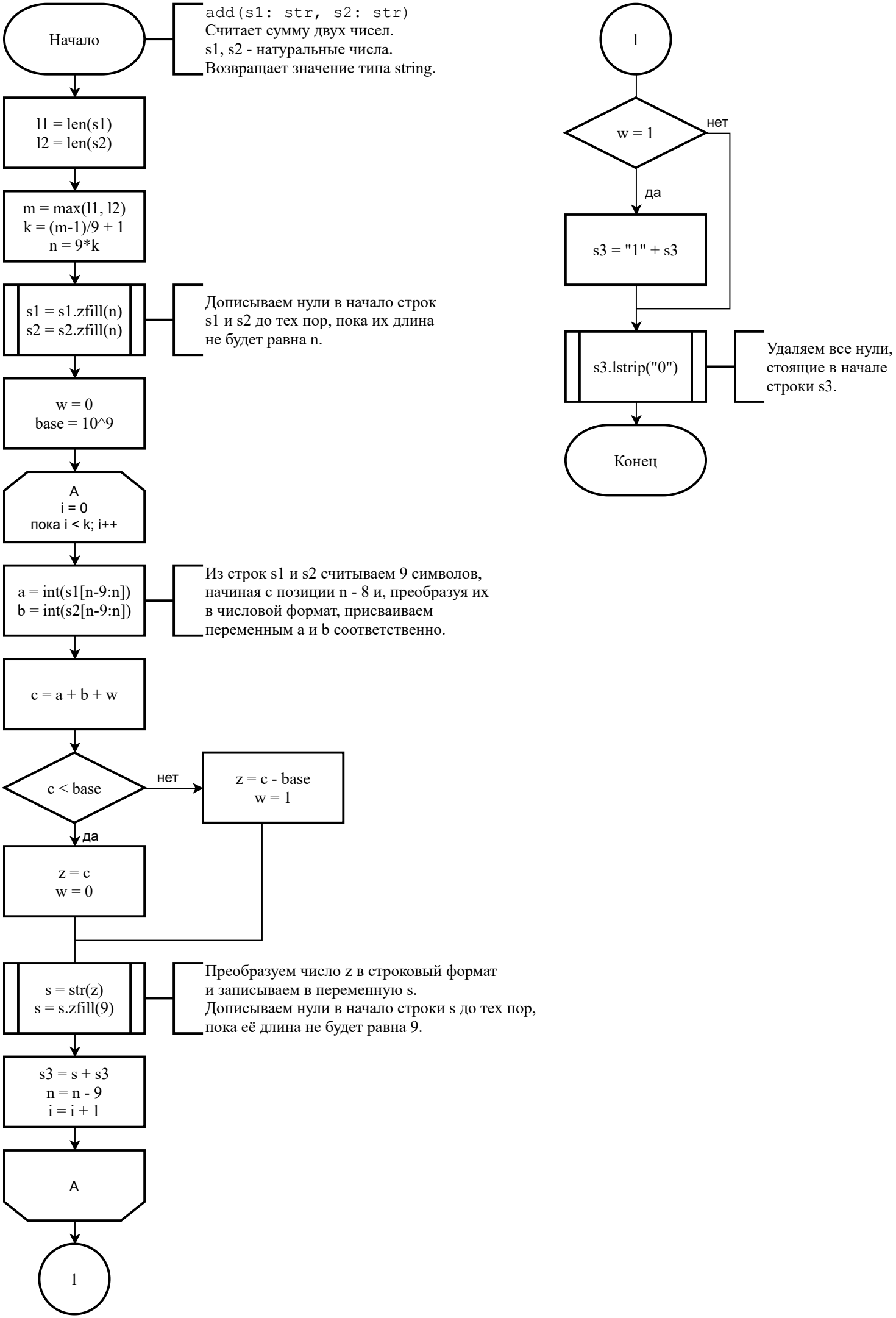
Вычисление обратных элементов реализовано в методе *BigInt.ring_ inv*. Для вычисления обратных элементов используется алгоритм описанный в [1].

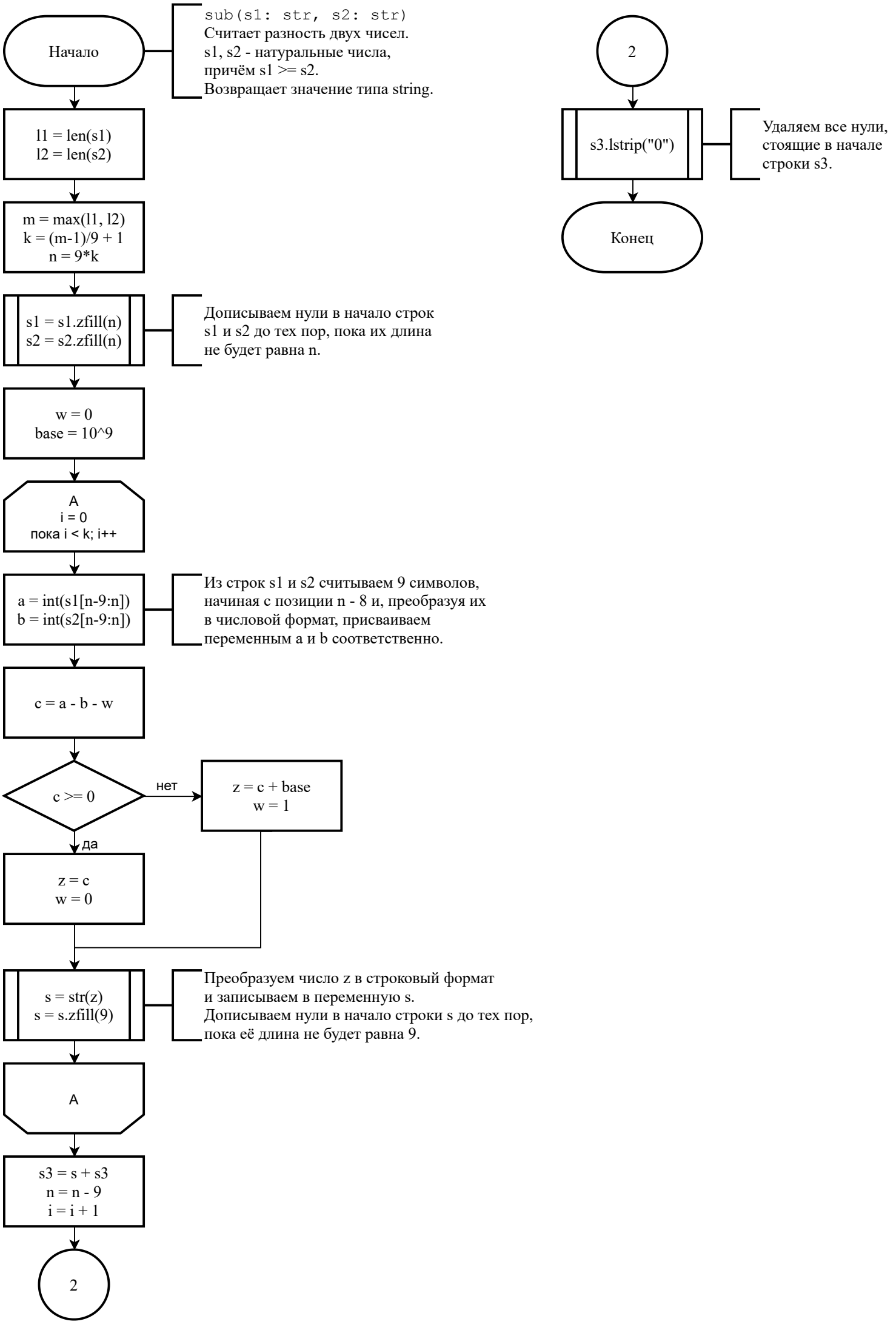
4.12 Возведение в степень в кольце вычетов

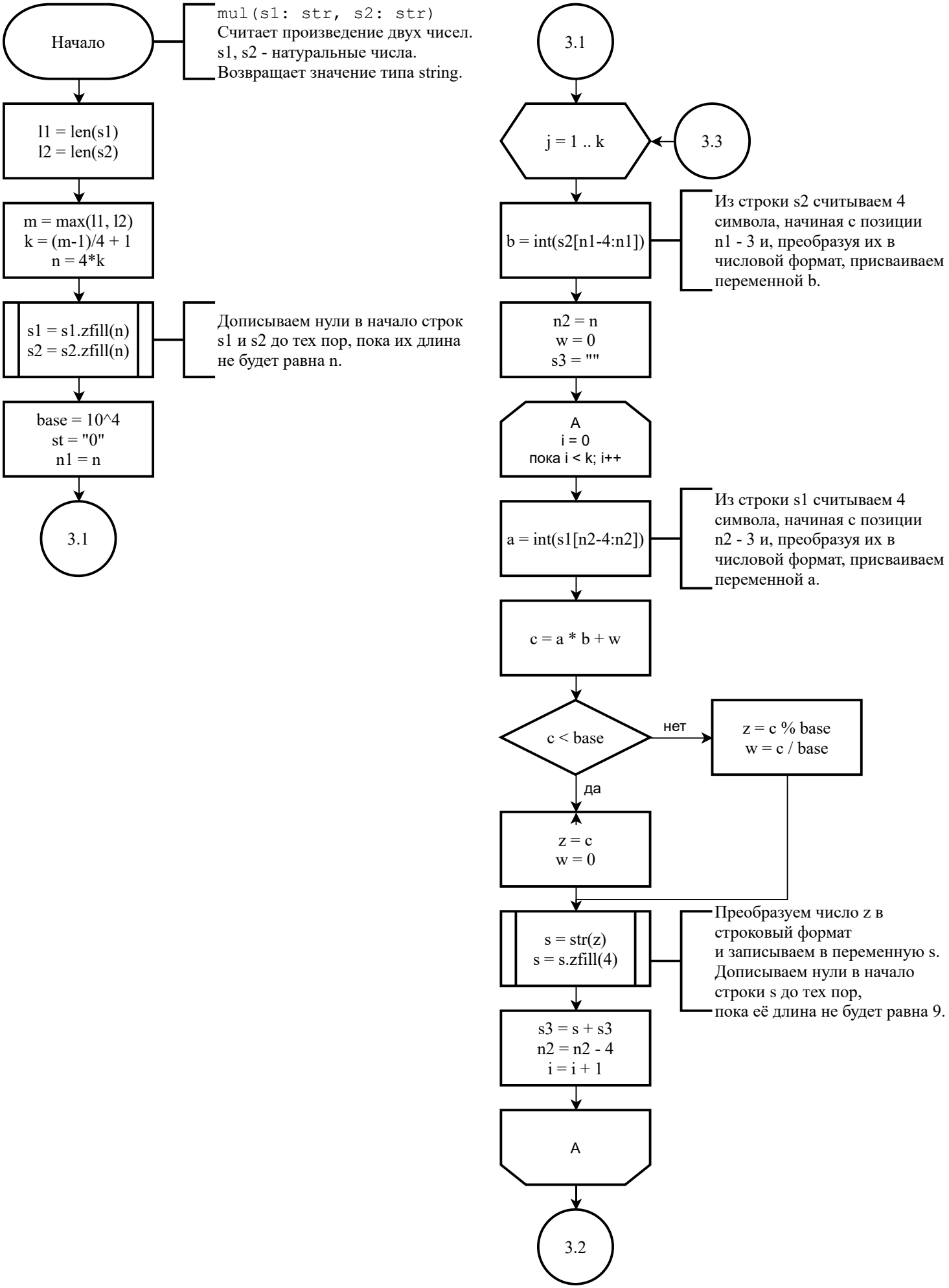
Возведение в степень реализовано в методе *BigInt.ring_ pow*. Для возведения в степень используется алгоритм описанный в [1].

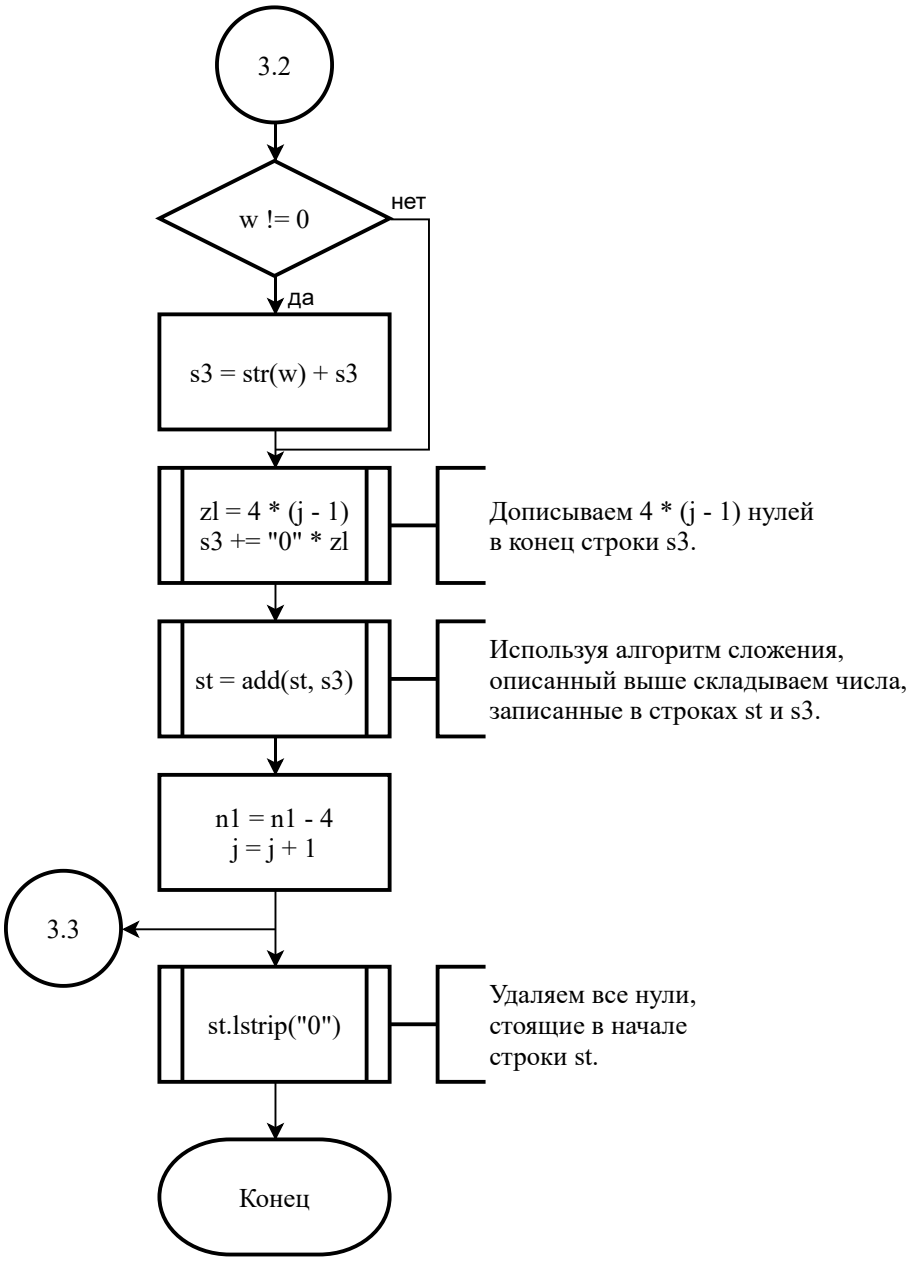
5 Блок-схемы алгоритмов

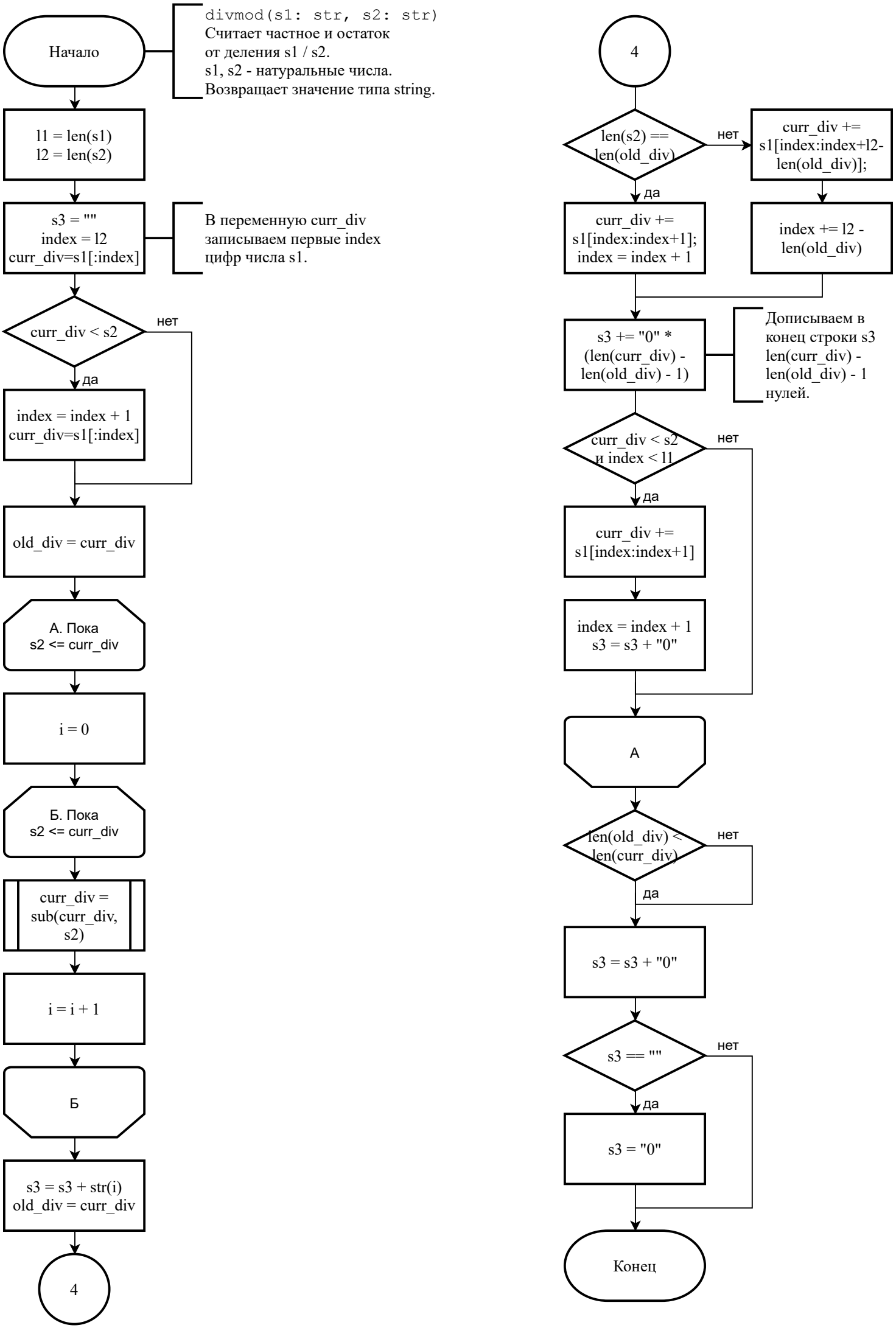
Ниже представлены блок-схемы функций, используемых для работы с длинными числами.

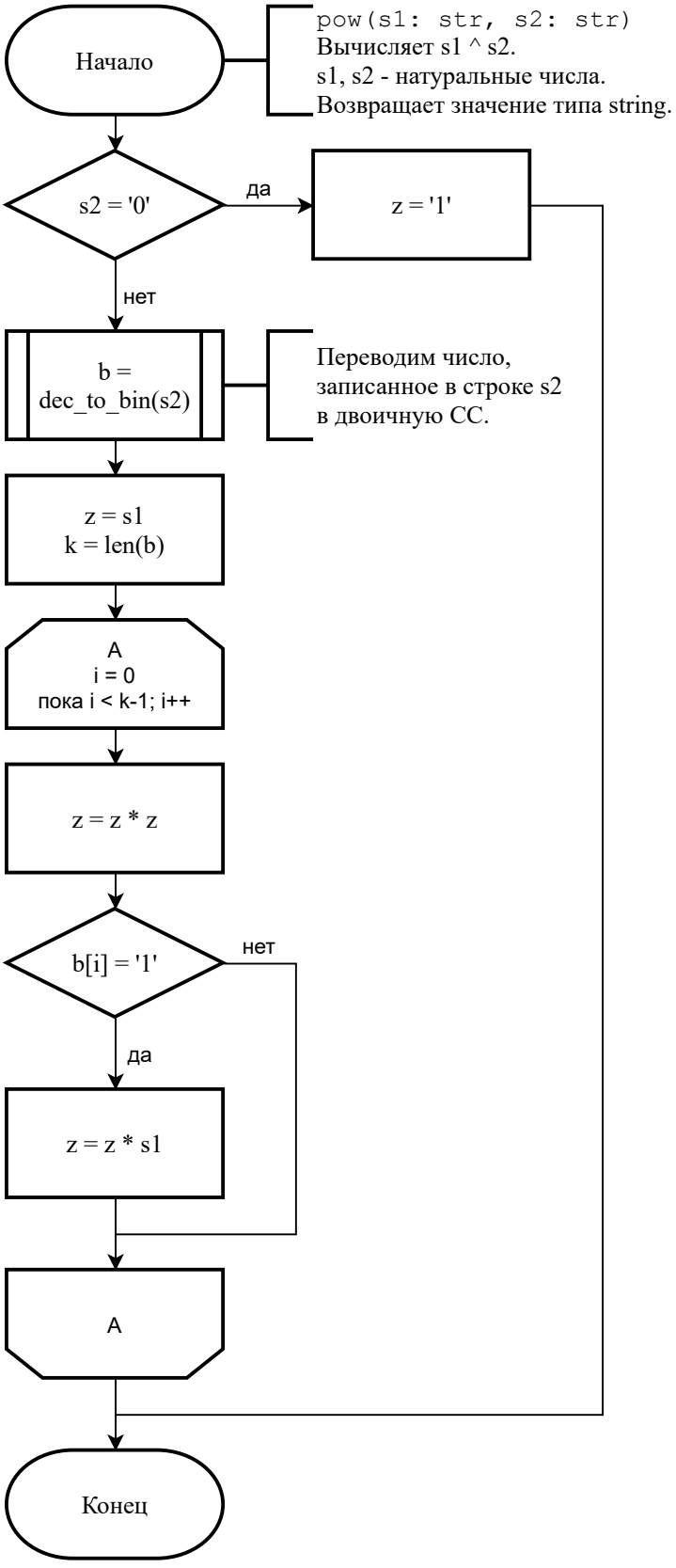

Начало
add(s1: str, s2: str)
Считает сумму двух чисел.
s1, s2 - натуральные числа.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/9 + 1
n = 9*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк s1 и s2 до тех пор, пока их длина не будет равна n.
w = 0
base = 10^9
A
i = 0
пока i < k; i++
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
A
1
1
w = 1
нет
да
s3 = "1" + s3
s3.lstrip("0")
Удаляем все нули, стоящие в начале строки s3.
Конец

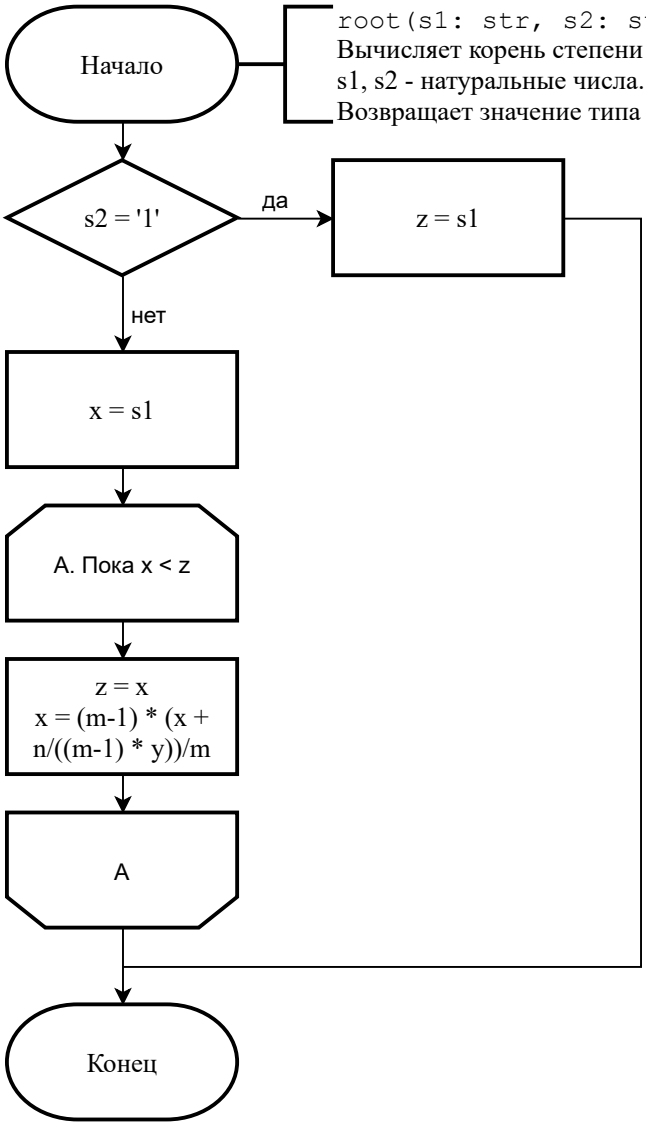

Начало
sub(s1: str, s2: str)
Считает разность двух чисел.
s1, s2 - натуральные числа,
причём s1 >= s2.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/9 + 1
n = 9*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк
s1 и s2 до тех пор, пока их длина
не будет равна n.
w = 0
base = 10^9
A
i = 0
пока i < k; i++
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.
A
s3 = s + s3
n = n - 9
i = i + 1
2
2
s3.lstrip("0")
Удаляем все нули,
стоящие в начале
строки s3.
Конец

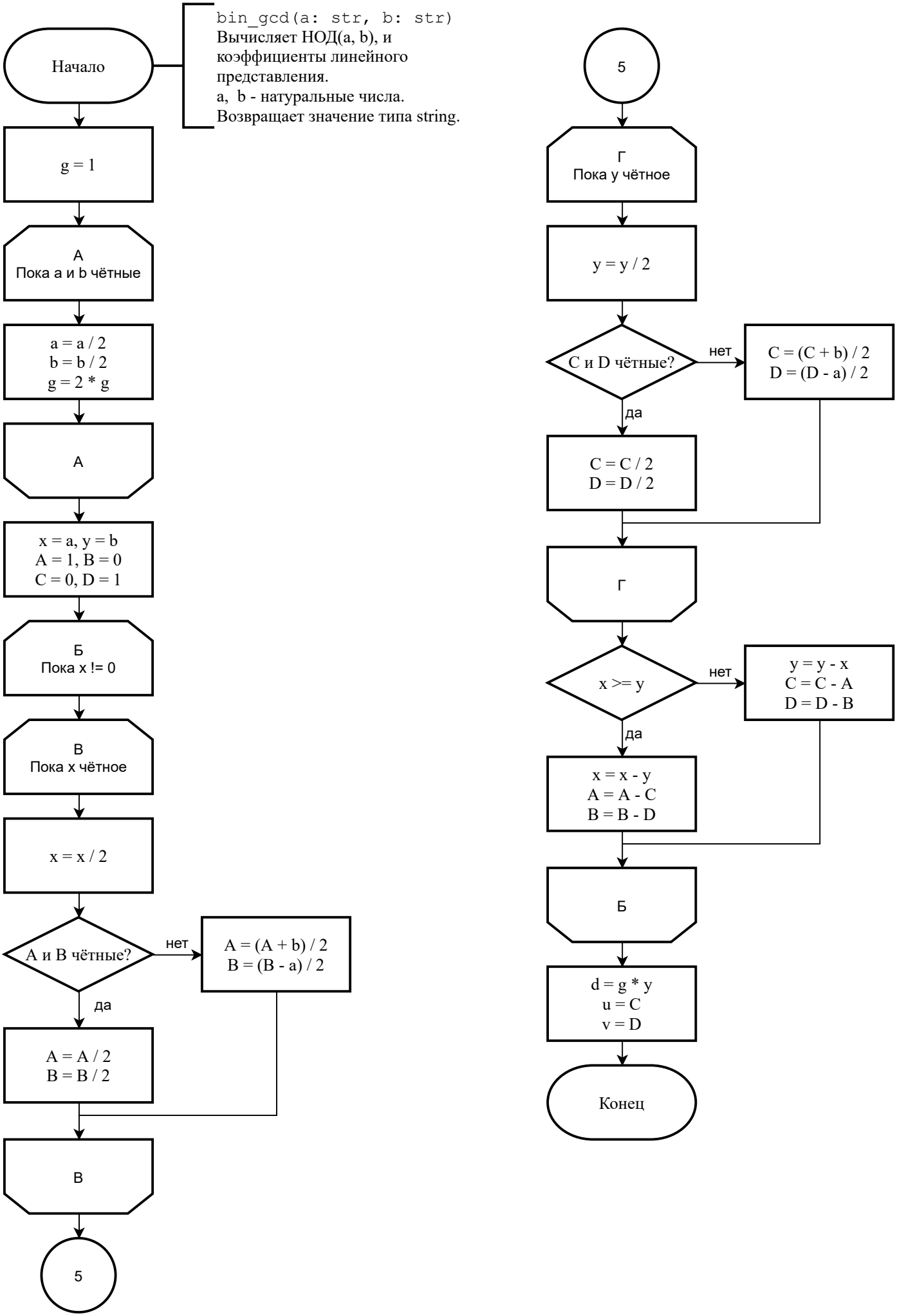

Начало
mul(s1: str, s2: str)
Считает произведение двух чисел.
s1, s2 - натуральные числа.
Возвращает значение типа string.
l1 = len(s1)
l2 = len(s2)
m = max(l1, l2)
k = (m-1)/4 + 1
n = 4*k
s1 = s1.zfill(n)
s2 = s2.zfill(n)
Дописываем нули в начало строк s1 и s2 до тех пор, пока их длина не будет равна 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 = ""
A
i = 0
пока i < k; i++
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
A
3.2

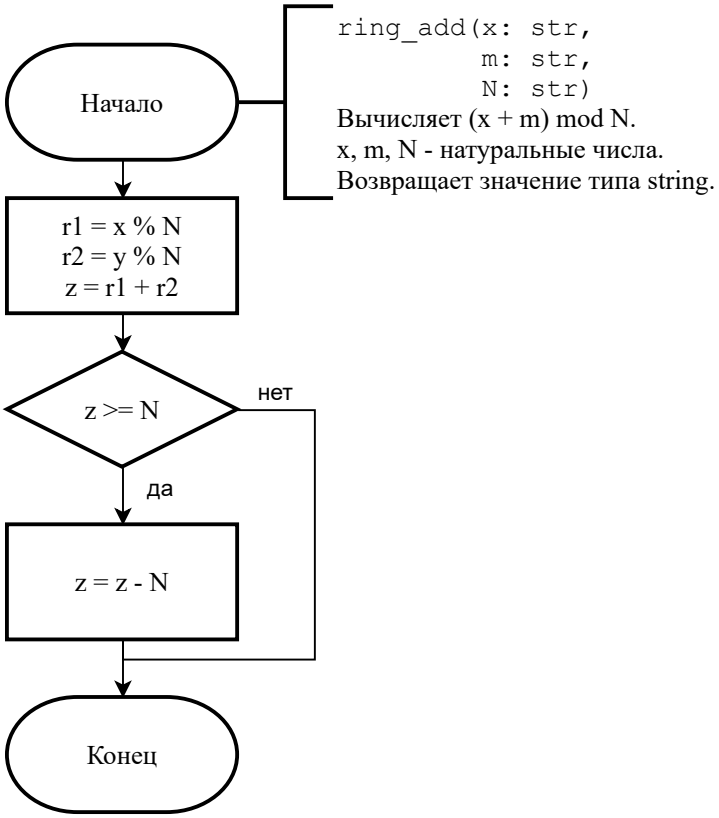

3.2
w != 0
нет
да
s3 = str(w) + s3
zl = 4 * (j - 1)
s3 += "0" * zl
Дописываем 4 * (j - 1) нулей
в конец строки s3.
st = add(st, s3)
Используя алгоритм сложения,
описанный выше складываем числа,
записанные в строках st и s3.
n1 = n1 - 4
j = j + 1
3.3
st.lstrip("0")
Удаляем все нули,
стоящие в начале
строки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


Начало
pow(s1: str, s2: str)
Вычисляет s1 ^ s2.
s1, s2 - натуральные числа.
Возвращает значение типа string.
s2 = '0'
да
z = '1'
нет
b = dec_to_bin(s2)
Переводим число, записанное в строке s2 в двоичную СС.
z = s1
k = len(b)
A
i = 0
пока i < k-1; i++
z = z * z
b[i] = '1'
нет
да
z = z * s1
A
Конец

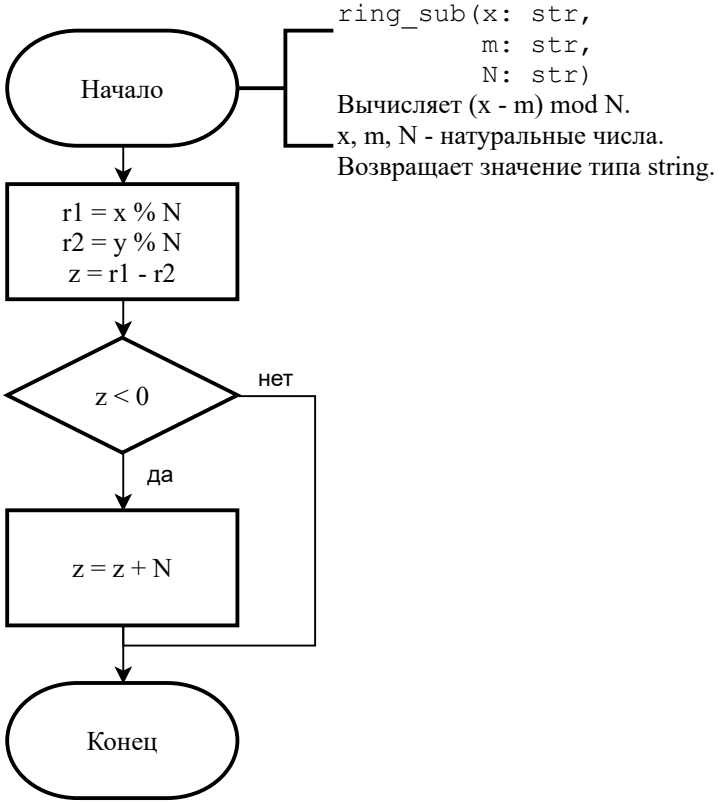

Начало
root(s1: str, s2: str)
Вычисляет корень степени s2 из s1.
s1, s2 - натуральные числа.
Возвращает значение типа string.
s2 = '1'
да
z = s1
нет
x = s1
A. Пока x < z
z = x
x = (m-1) * (x +
n/((m-1) * y))/m
A
Конец


bin_gcd(a: str, b: str)
Вычисляет НОД(a, b), и коэффициенты линейного представления.
a, b - натуральные числа.
Возвращает значение типа string.
Начало
g = 1
А
Пока a и b чётные
a = a / 2
b = b / 2
g = 2 * g
А
x = a, y = b
A = 1, B = 0
C = 0, D = 1
Б
Пока x != 0
В
Пока x чётное
x = x / 2
A и B чётные?
нет
A = (A + b) / 2
B = (B - a) / 2
да
A = A / 2
B = B / 2
В
5
5
Г
Пока y чётное
y = y / 2
C и D чётные?
нет
C = (C + b) / 2
D = (D - a) / 2
да
C = C / 2
D = D / 2
Г
x >= y
нет
y = y - x
C = C - A
D = D - B
да
x = x - y
A = A - C
B = B - D
Б
d = g * y
u = C
v = D
Конец



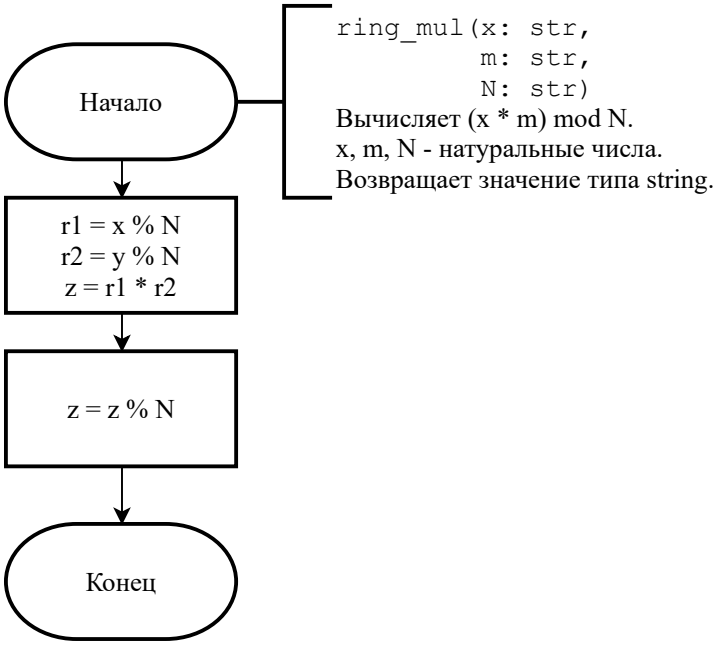

```
ring_add(x: str,
         m: str,
         N: str)
```

Вычисляет (x + m) mod N.
x, m, N - натуральные числа.
Возвращает значение типа string.



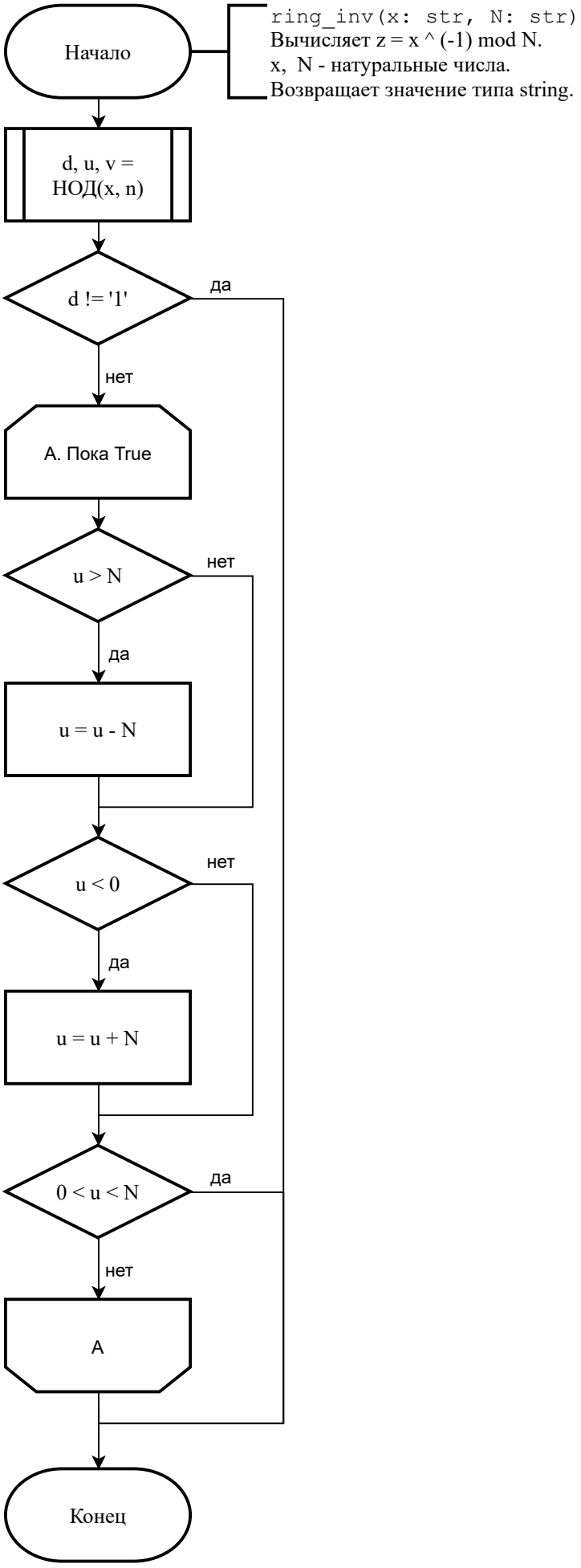

```
ring_sub(x: str,
         m: str,
         N: str)
```

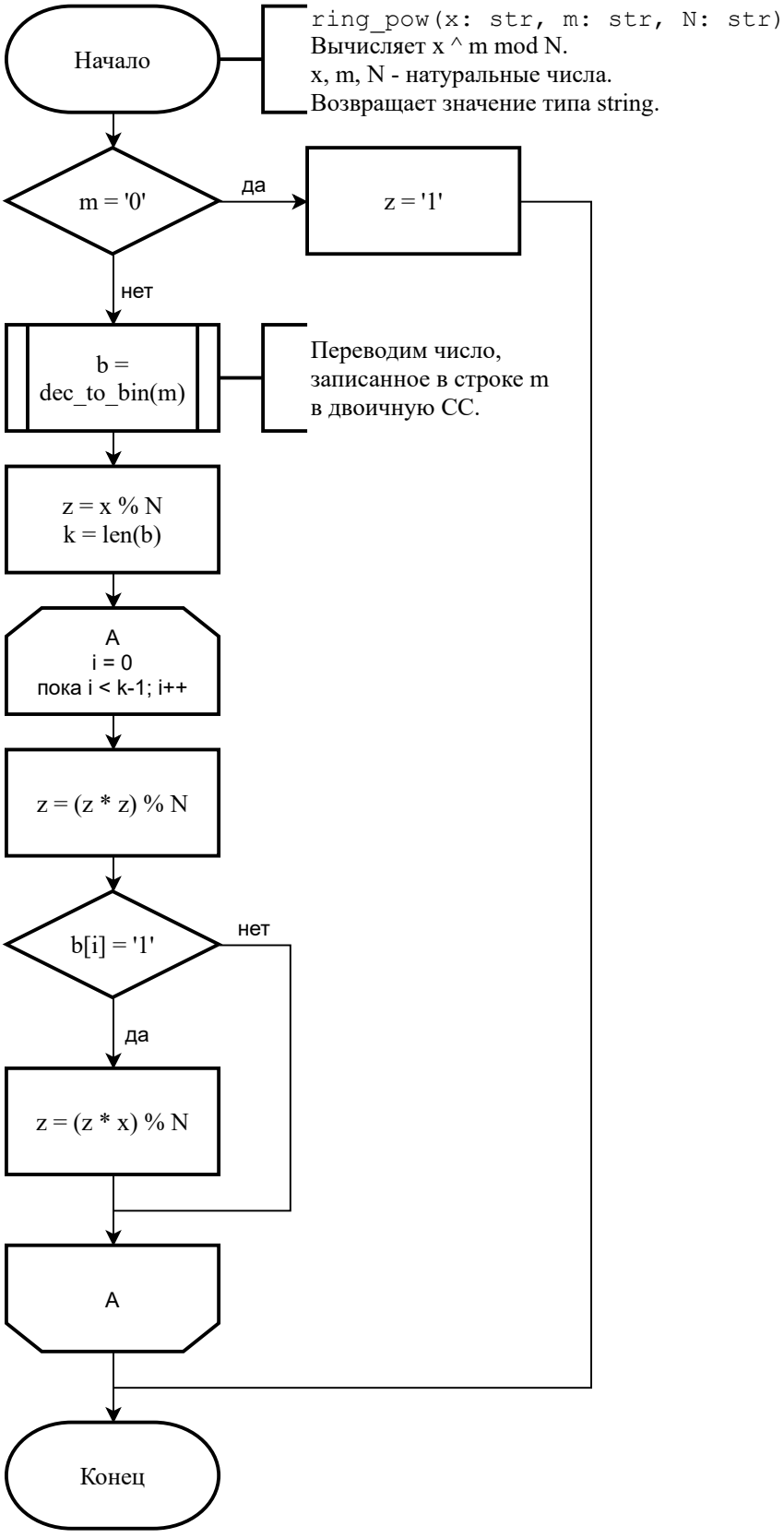
Вычисляет (x - m) mod N.
x, m, N - натуральные числа.
Возвращает значение типа string.




```
ring_mul(x: str,
         m: str,
         N: str)
```

Вычисляет (x * m) mod N.
x, m, N - натуральные числа.
Возвращает значение типа string.


Начало
ring_inv(x: str, N: str)
Вычисляет z = x ^ (-1) mod N.
x, N - натуральные числа.
Возвращает значение типа string.
d, u, v =
НОД(x, n)
d != '1'
да
нет
A. Пока True
u > N
нет
да
u = u - N
u < 0
нет
да
u = u + N
0 < u < N
да
нет
A
Конец


Начало
ring_pow(x: str, m: str, N: str)
Вычисляет x ^ m mod N.
x, m, N - натуральные числа.
Возвращает значение типа string.
m = '0'
да
z = '1'
нет
b = dec_to_bin(m)
Переводим число, записанное в строке m в двоичную СС.
z = x % N
k = len(b)
A
i = 0
пока i < k-1; i++
z = (z * z) % N
b[i] = '1'
нет
да
z = (z * x) % N
A
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
- $x1^{(-1)}modN$ = 5264157999473642
- $x2^{y2}modN$ = 12182065501559763

7 Список литературы

1. *Завгородний М. Г.*, *Майорова С. П.* Программирование. Криптографические алгоритмы: учебное пособие. — Воронеж : Издательский дом ВГУ, 2018.

2. 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, 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 mod y',
300         '6) x ^ y',
301         '7) КореньизX степениY',
302         '8) НОД(x, y)',
303         '9) (x + y) mod N',
304         '10) (x - y) mod N',
305         '11) (x * y) mod N',
306         '12) x^(-1) mod N',
307         '13) x^y mod N',
308     ])
309     print(menu_text)
310     choice = input('Введите номероперации: ')
311 
312     if int(choice) < 1 or int(choice) > 13:
313         print('Выбрано несуществующеезначение:(')
314         input('Для выходанажмитеEnter...')
315         exit(0)
316 
317     x = BigInt(input('Введите первоечисло(x): '))
318     if choice != '12':
319         y = BigInt(input('Введите второечисло(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 элементчисла{x} помодулю{n} несуществует!')
349         else:
350             print('x^(-1) mod N =', BigInt.ring_inv(x, n))
351     elif choice == '13':
352         print('x^y mod N =', BigInt.ring_pow(x, y, n))
353 
354     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_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()
```