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 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.

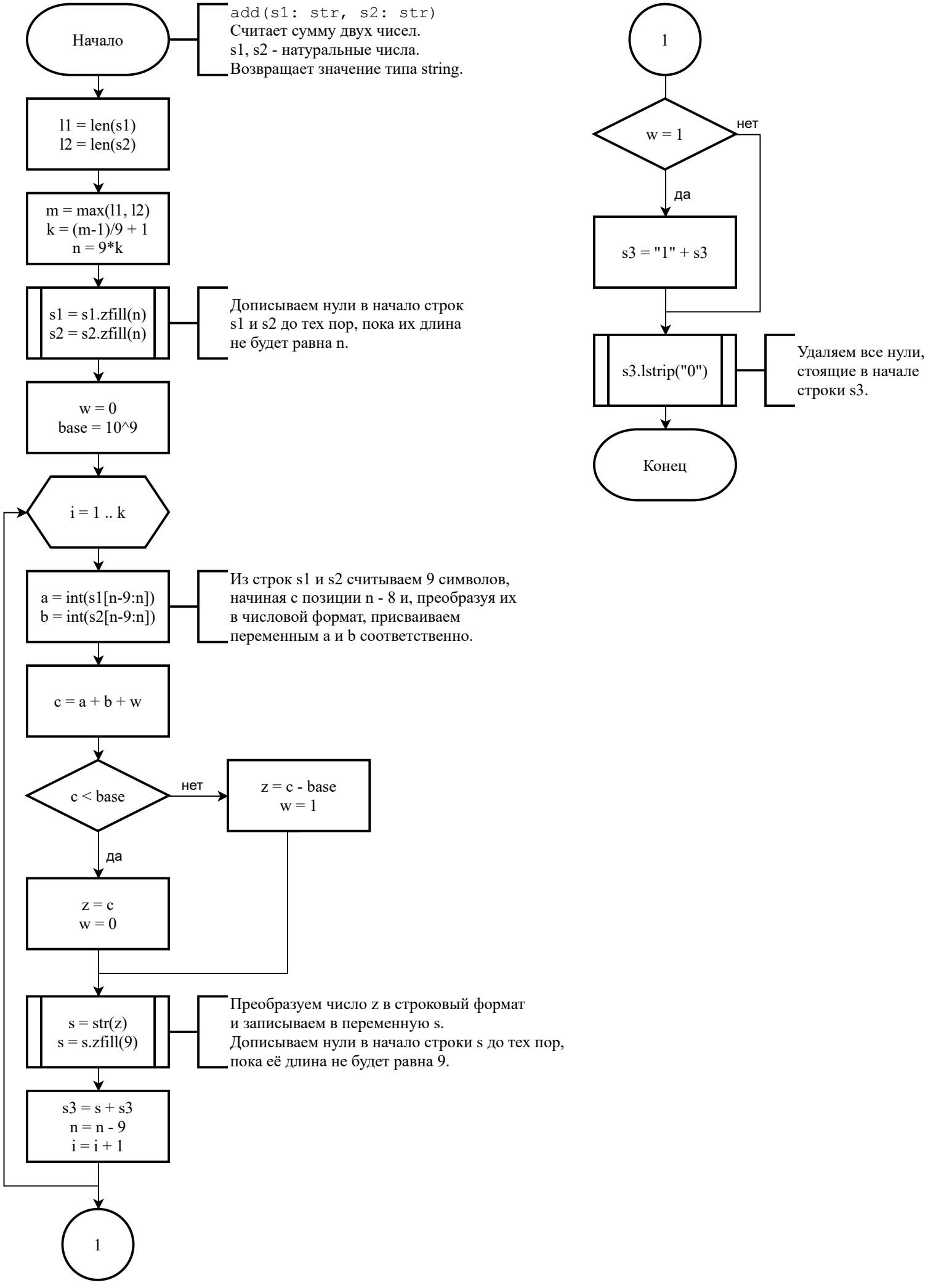
(h) Полагаем $n1 = n1$ - 4 и $j = j + 1$

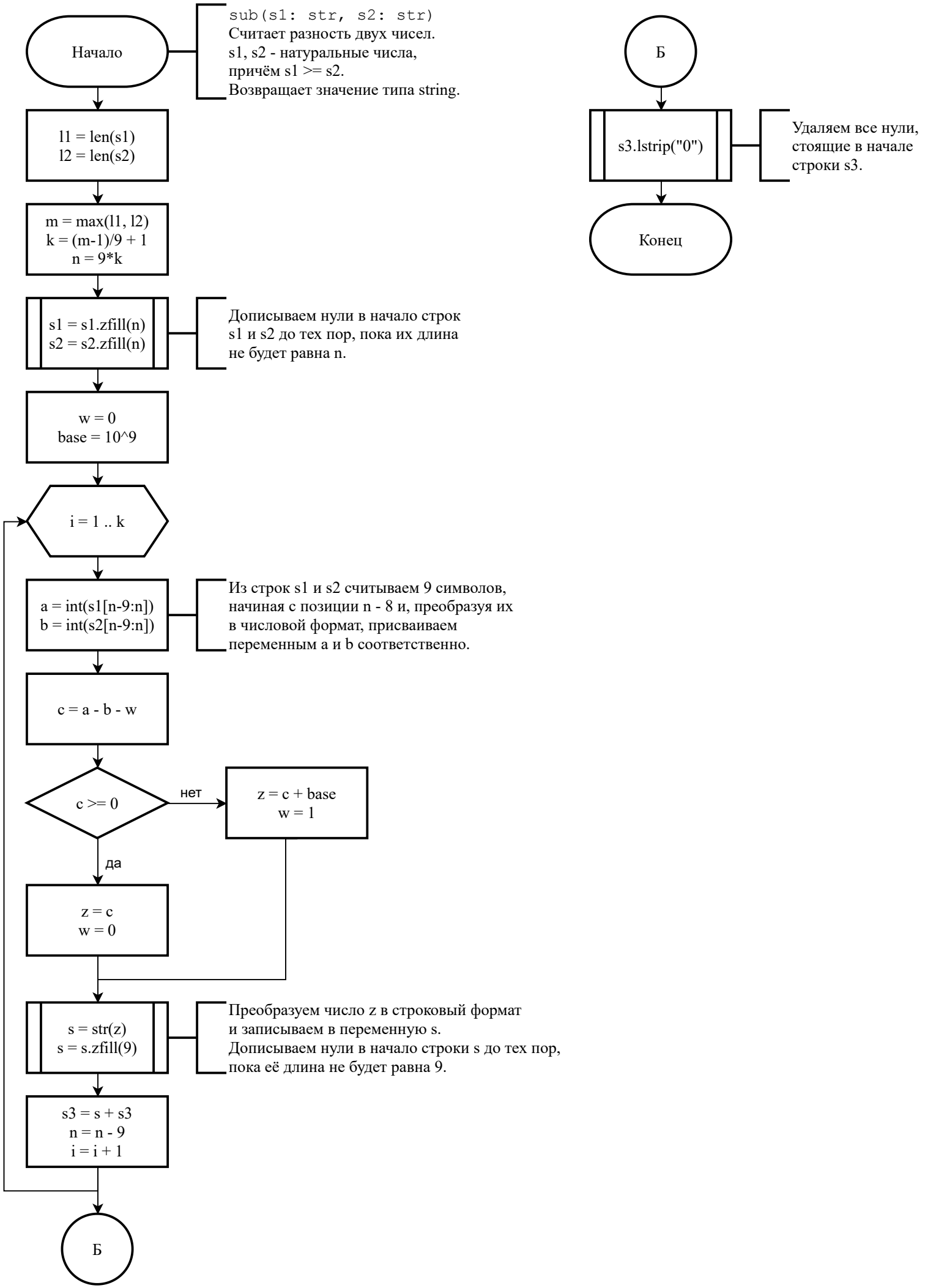
4.4 Деление с остатком

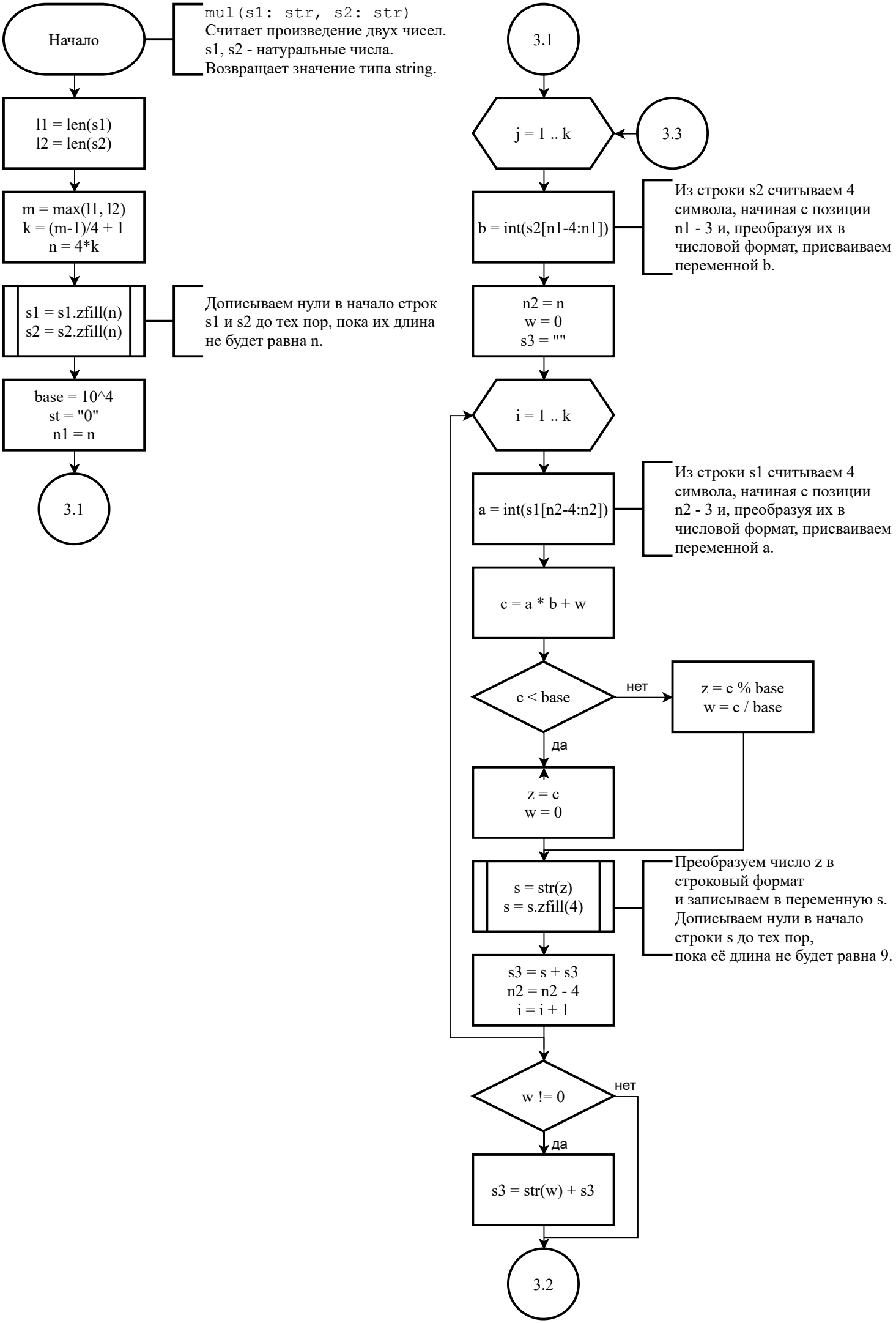
Деление с остатком реализовано в функции l_divmod. В качестве алгоритма используется деление в столбик с небольшими модификациями для ускорения работы и сокращения количества итераций.

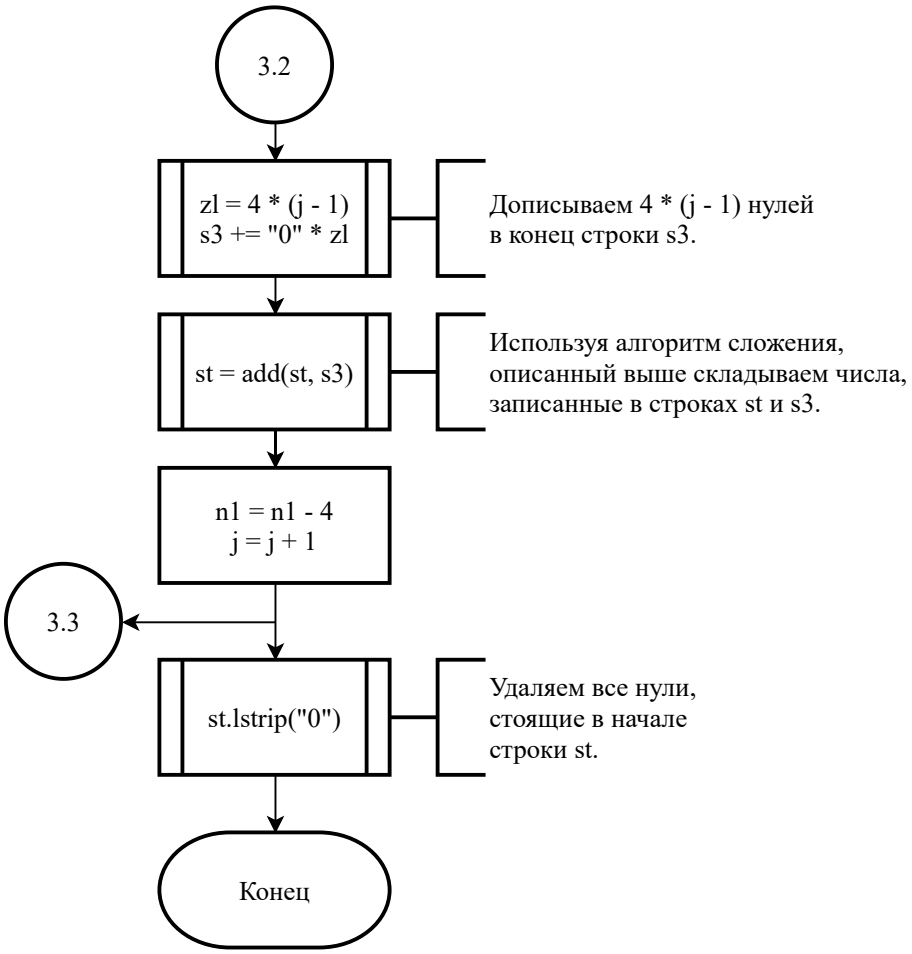
4.5 Блок-схемы алгоритмов

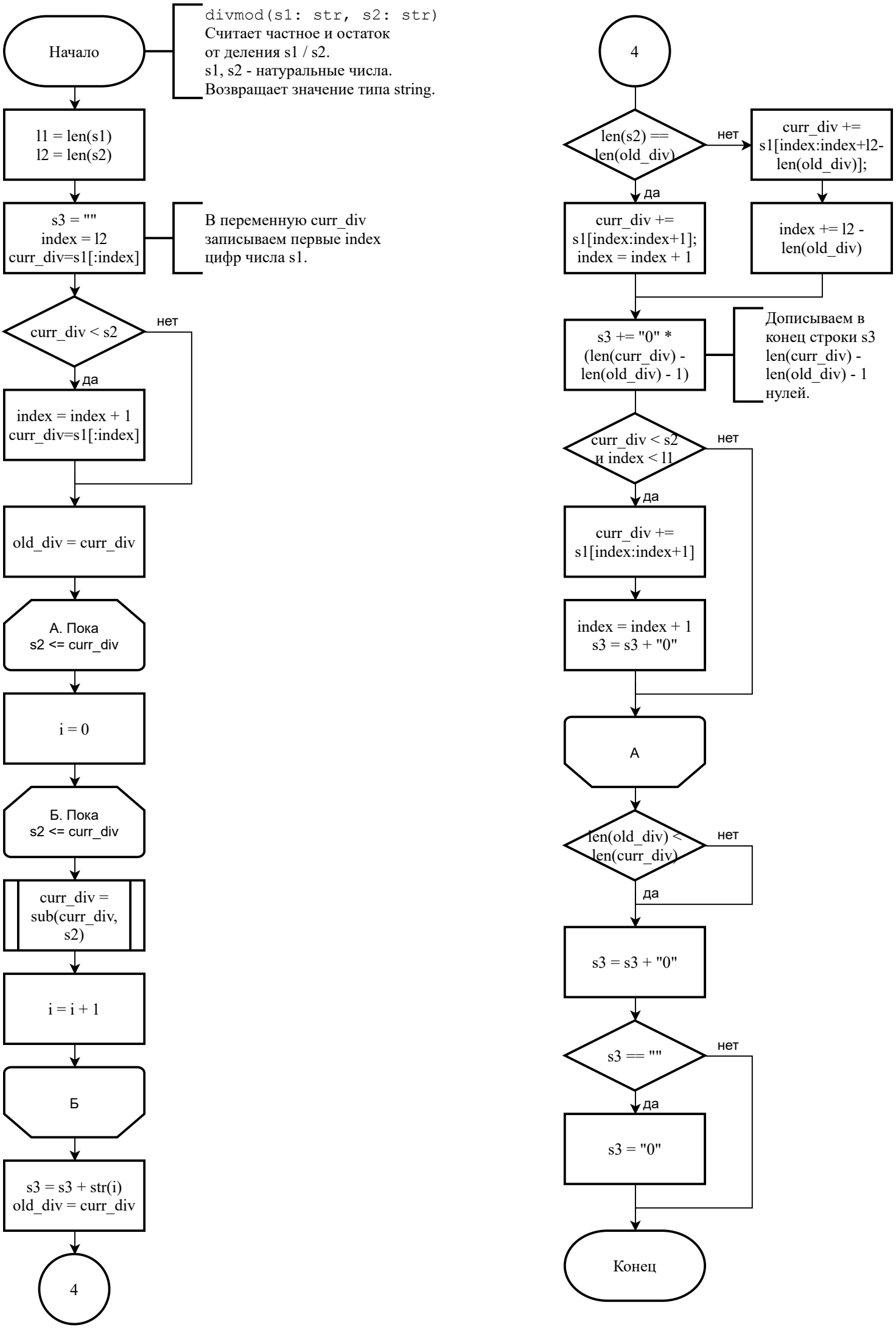
Ниже представлены блок-схемы функций, используемых для работы с длинными числами.


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

5 Примеры работы программы

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

```
1  from bigint import BigInt
2
3  x = BigInt('-99999999999999999999')
4  y = BigInt('11111111111111111111')
5
6  print('x =', x)
7  print('y =', y)
8
9  print('x + y =', x + y)
10 print('x - y =', x - y)
11 print('x * y =', x * y)
12 print('x / y =', x / y)
13 print('x mod y =', x % y)
```

Вывод программы:

- x = -99999999999999999999
- y = 11111111111111111111
- x + y = -88888888888888888888
- x - y = -111111111111111111110
- x * y = -1111111111111111111088888888888888888889
- x / y = -9
- x mod y = 0

6 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/`.

7 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.

7.1 bigint.py

```
1  from long_math import l_add, l_divmod, l_mul, 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
119 if __name__ == '__main__':
120     x = BigInt(input('Введите первоечисло(x): '))
121     y = BigInt(input('Введите второечисло(y): '))
122     menu_text = '\n'.join([
123         'Выберите действие:',
124         '1) x + y',
125         '2) x - y',
126         '3) x * y',
127         '4) x / y',
128         '5) x mod y'
129     ])
130     print(menu_text)
131     choice = input()
132     if choice == '1':
133         print('x + y =', x + y)
134     elif choice == '2':
135         print('x - y =', x - y)
136     elif choice == '3':
137         print('x * y =', x * y)
138     elif choice == '4':
139         print('x / y =', x / y)
140     elif choice == '5':
141         print('x mod y =', x % y)
142     else:
143         print('Выбрано несуществующеезначение:(')
144
145     input('Для выходанажмитеEnter...')
```

7.2 tests.py

```
import unittest
from random import randint

from bigint import BigInt
from long_math import l_add, l_divmod, l_mul, 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))

    def test_less_than(self):
        for _ in range(self.TESTS_COUNT):
```

```
41              x = randint(self.MIN, self.MAX)
42              y = randint(self.MIN, self.MAX)
43              self.assertEqual(x < y, less_than(str(x), str(y)))
44
45
46  class TestBigInt(unittest.TestCase):
47
48      MIN = -10 ** 30
49      MAX = 10 ** 30
50      TESTS_COUNT = 10 ** 5
51
52      def test_add(self):
53          for _ in range(self.TESTS_COUNT):
54              x = randint(self.MIN, self.MAX)
55              y = randint(self.MIN, self.MAX)
56              big_x = BigInt(str(x))
57              big_y = BigInt(str(y))
58              self.assertEqual(x + y, big_x + big_y)
59
60      def test_sub(self):
61          for _ in range(self.TESTS_COUNT):
62              x = randint(self.MIN, self.MAX)
63              y = randint(self.MIN, self.MAX)
64              big_x = BigInt(str(x))
65              big_y = BigInt(str(y))
66              self.assertEqual(x - y, big_x - big_y)
67
68      def test_mul(self):
69          for _ in range(self.TESTS_COUNT):
70              x = randint(self.MIN, self.MAX)
71              y = randint(self.MIN, self.MAX)
72              big_x = BigInt(str(x))
73              big_y = BigInt(str(y))
74              self.assertEqual(x * y, big_x * big_y)
75
76      def test_div(self):
77          for _ in range(self.TESTS_COUNT):
78              x = randint(self.MIN, self.MAX)
79              y = randint(self.MIN, self.MAX)
80              big_x = BigInt(str(x))
81              big_y = BigInt(str(y))
```

```
82              self.assertEqual(int(x / y), big_x / big_y)
83
84      def test_mod(self):
85          for _ in range(self.TESTS_COUNT):
86              x = randint(self.MIN, self.MAX)
87              y = randint(self.MIN, self.MAX)
88              big_x = BigInt(str(x))
89              big_y = BigInt(str(y))
90              self.assertEqual(x % y, big_x % big_y)
91
92
93  if __name__ == '__main__':
94      unittest.main()
```