

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>		
Обучающийся	_____		А.А. Уткин
	<i>подпись</i>		

Воронеж 2021

Содержание

1	Постановка задачи	3
2	Используемые инструменты	4
3	Общая структура библиотеки целых длинных чисел	5
4	Примеры работы библиотеки	7
5	Вывод	8
6	Блок-схема методов библиотеки	9
7	Исходный код программы	19
	Список литературы	36

1 Постановка задачи

Составить алгоритм и написать соответствующую ему программу, позволяющую:

- выполнять операции сложения, вычитания и умножения в кольце вычетов;
- находить элементы, обратные к элементам, взаимно простым с модулем кольца;
- возводить в натуральную степень элементы кольца вычетов.

2 Используемые инструменты

Для решения вышеуказанной задачи были использованы следующие инструменты:

- Основным ЯП был выбран Python версии 3.9.2;
- Для компиляции программы в бинарный файл .exe использован конвертер файлов Auto PY to EXE, который использует для своей работы PyInstaller.

3 Общая структура библиотеки целых длинных чисел

В библиотеке целых длинных содержится класс «BigInt», в котором объявлены следующие методы:

- Реализация алгоритма сложения в кольце вычетов.
Алгоритм был реализован на основе работы сложения и нахождения остатка длинных чисел. Изменения касались лишь тех моментов, когда необходимо было реализовать логику кольца вычетов.
- Реализация алгоритма вычитания в кольце вычетов.
Алгоритм был реализован на основе работы вычитания и нахождения остатка длинных чисел. Изменения касались лишь тех моментов, когда необходимо было реализовать логику кольца вычетов.
- Реализация алгоритма умножения в кольце вычетов.
Алгоритм был реализован на основе работы умножения и нахождения остатка длинных чисел. Изменения касались лишь тех моментов, когда необходимо было реализовать логику кольца вычетов.
- Реализация алгоритма возведения в степень в кольце вычетов.
Алгоритм был реализован на основе работы бинарном алгоритме возведения в степень и нахождения остатка длинных чисел. Изменения касались лишь тех моментов, когда необходимо было реализовать логику кольца вычетов.
- Реализация расширенного алгоритма Евклида для поиска НОД двух длинных целых чисел, а также коэффициентов.
Данный алгоритм был реализован для его дальнейшего использования в алгоритме нахождения обратного элемента в кольце вычетов. Этот алгоритм, в отличие от других реализаций алгоритмов поиска НОД, является методом класса, а не функцией.
- Реализация нахождения обратного элемента в кольце вычетов.
Алгоритм был реализован на основе работы сложения, вычитания и расширенного алгоритма Евклида для поиска НОД длинных чисел. В

случае использования расширенного алгоритма Евклида для поиска НОД от данного метода требуются коэффициенты.

Класс «BigInt» содержит в себе два основных поля:

1. Поле хранения числа «value».

Представляет собой переменную типа строка, в котором содержится число экземпляра класса;

2. Поле хранения знака числа «is_neg».

Представляет собой переменную типа bool, в которой содержится информация о знаке числа. Значение True эквивалентно отрицательному числу, значение False - положительному;

Создания экземпляра класса «BigInt» происходит следующие способами:

- Создание экземпляра класса без передачи аргументов. Числовое значение такого экземпляра будет равно нулю.

```
1 a = BigInt()
```

- Создание экземпляра класса с передачей в аргумент строки, которая может валидно быть приведена к типу целого числа.

```
1 a = BigInt('-1234567890') # a = -1234567890
2 b = BigInt('1234567890') # b = 1234567890
3 d = BigInt('0') # d = 0
```

- Создание экземпляра класса с передачей в аргумент целого числа.

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(1234567890) # b = 1234567890
3 d = BigInt(0) # d = 0
```

- Создание экземпляра класса с передачей в аргумент экземпляра класса «BigInt».

```
1 a = BigInt(-1234567890) # a = -1234567890
2 b = BigInt(a) # b = -1234567890
```

4 Примеры работы библиотеки

В качестве примера работы будут использоваться прямые вызовы методов класса «BigInt».

Пусть даны:

- два длинных числа $x1$ и $y1$, сохраненных в экземпляр класса «BigInt»;
- два обычных длинных числа $x2$ и $y2$, для степенных операций;
- длинное число N , которое будет использовано в качестве модуля.

```
1 x1 = BigInt('9999999999999999999')
2 y1 = BigInt('1111111111111111111')
3 x2 = BigInt('99')
4 y2 = BigInt('11')
5 N = BigInt('12345678998765432')
```

- Выполняем сложение в кольце вычетов:

```
1 print(BigInt.ring_add(x1, y1))
```

Вывод: 12222223110

- Выполняем вычитание в кольце вычетов:

```
1 print(BigInt.ring_sub(x1, y1))
```

Вывод: 97777778488

- Выполняем умножение в кольце вычетов:

```
1 print(BigInt.ring_mul(x1, y1))
```

Вывод: 2101000081689

- Выполняем нахождение обратного элемента в кольце вычетов:

```
1 print(BigInt.ring_inv_el(x1, N))
```

Вывод: 5264157999473642

- Выполняем возведение в степень в кольце вычетов:

```
1 print(BigInt.ring_pow(x2, y2))
```

Вывод: 12182065501559763

5 Вывод

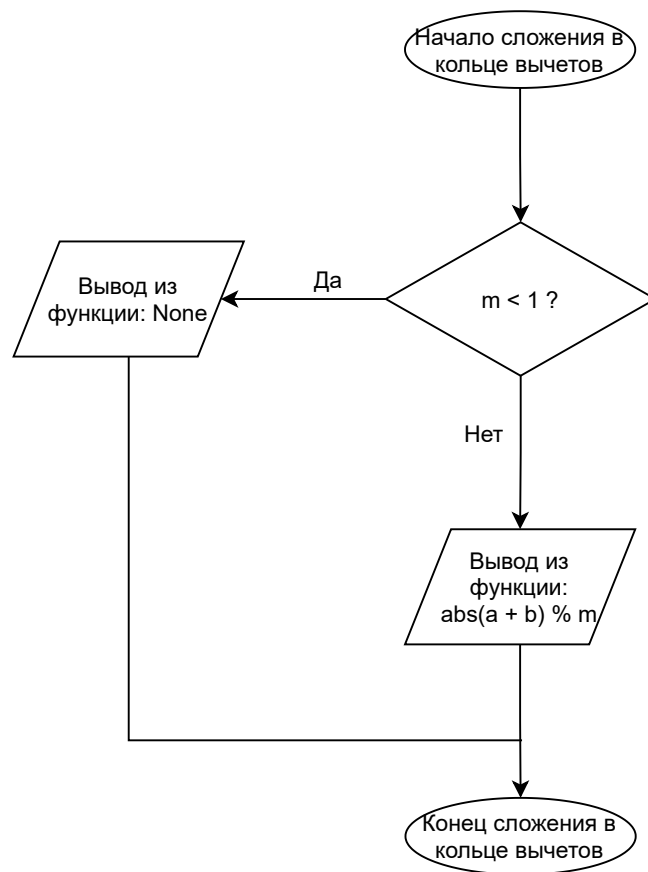
Мною был составлен алгоритм (в виде блок-схемы) и написана на языке Python программа, позволяющая:

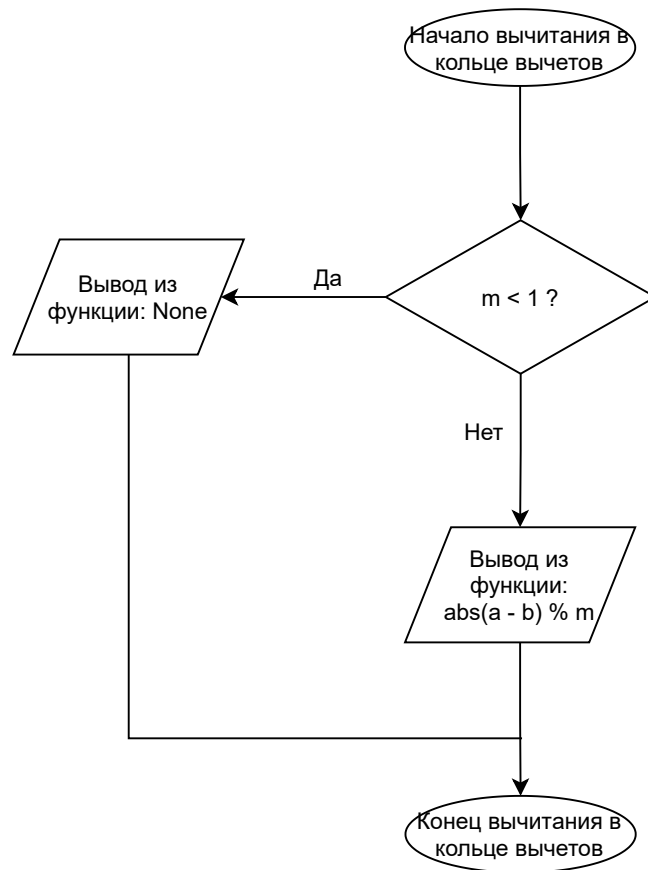
- производить сложение в кольце вычетов двух длинных чисел;
- производить вычитания в кольце вычетов двух длинных чисел;
- производить умножения в кольце вычетов двух длинных чисел;
- производить возведение в степень числа в кольце вычетов;
- находить НОД двух длинных чисел, а также коэффициенты;
- находить обратный элемент в кольце вычетов;

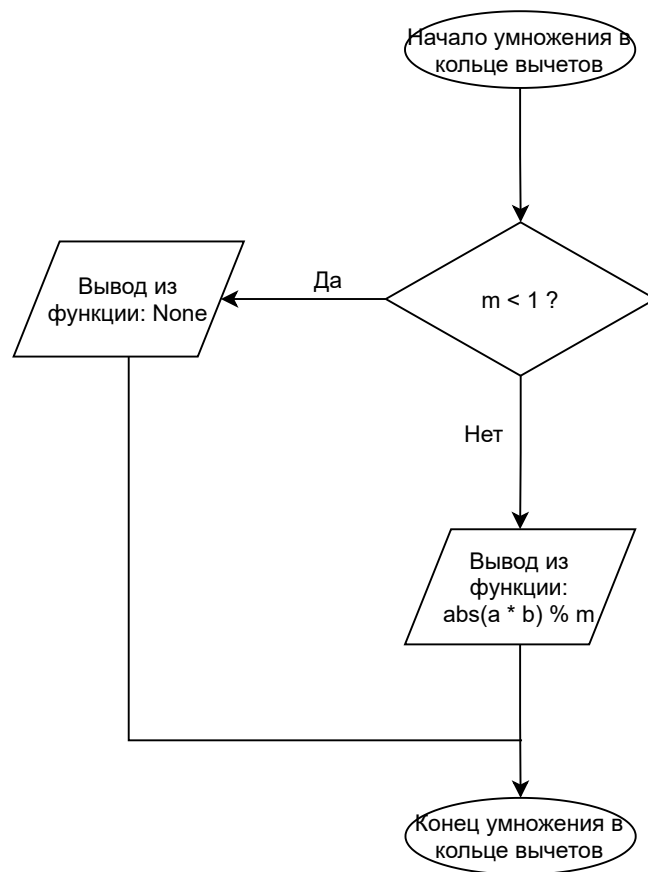
6 Блок-схема методов библиотеки

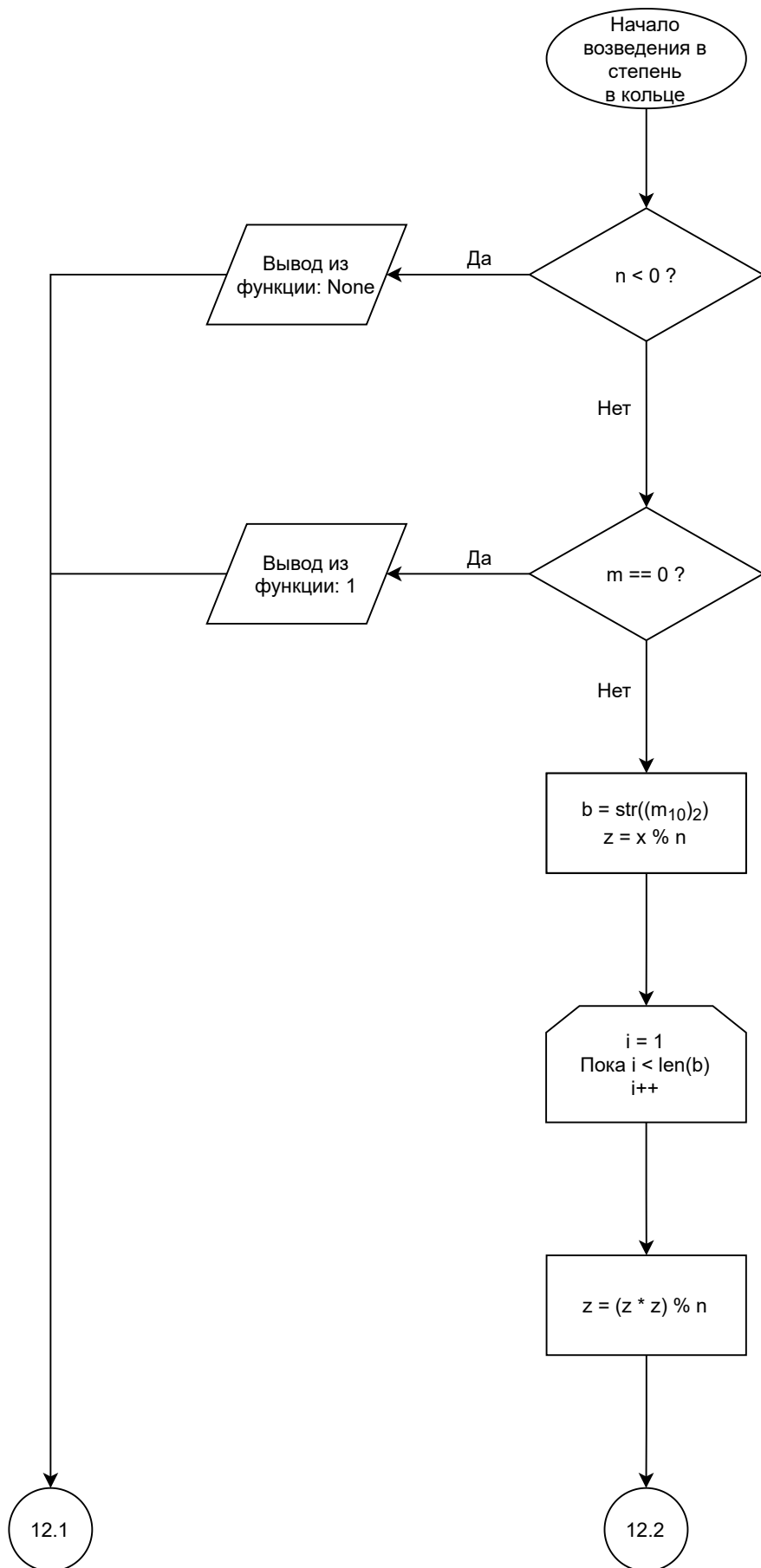
Ниже представлены блок-схемы методов в следующем порядке:

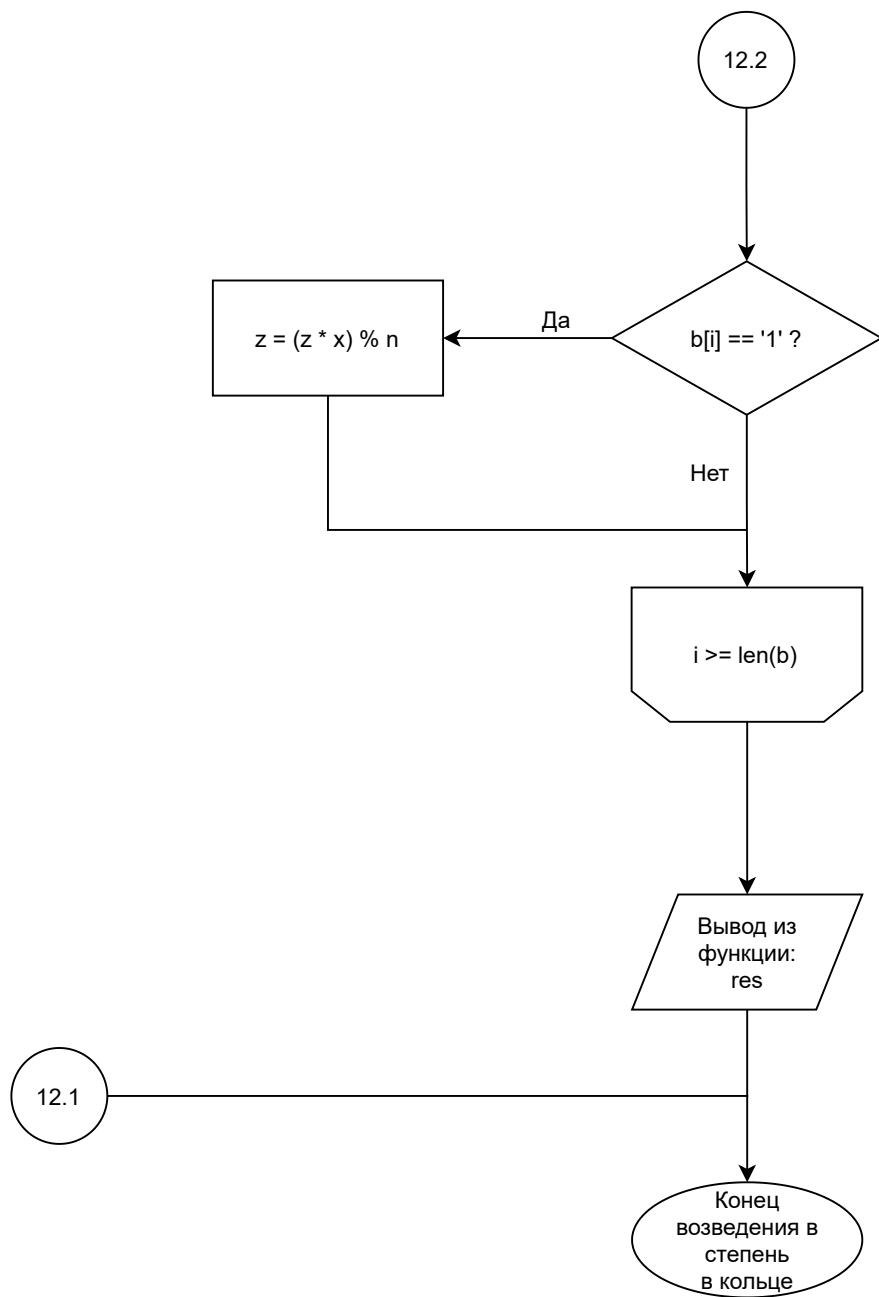
1. Метод нахождения суммы в кольце вычетов двух длинных чисел;
2. Метод нахождения разности в кольце вычетов двух длинных чисел;
3. Метод нахождения произведения в кольце вычетов двух длинных чисел;
4. Метод, позволяющий возводить в степень числа в кольце вычетов;
5. Метод нахождения НОД двух длинных чисел, а также коэффициенты;
6. Метод нахождения обратного элемента в кольце вычетов.

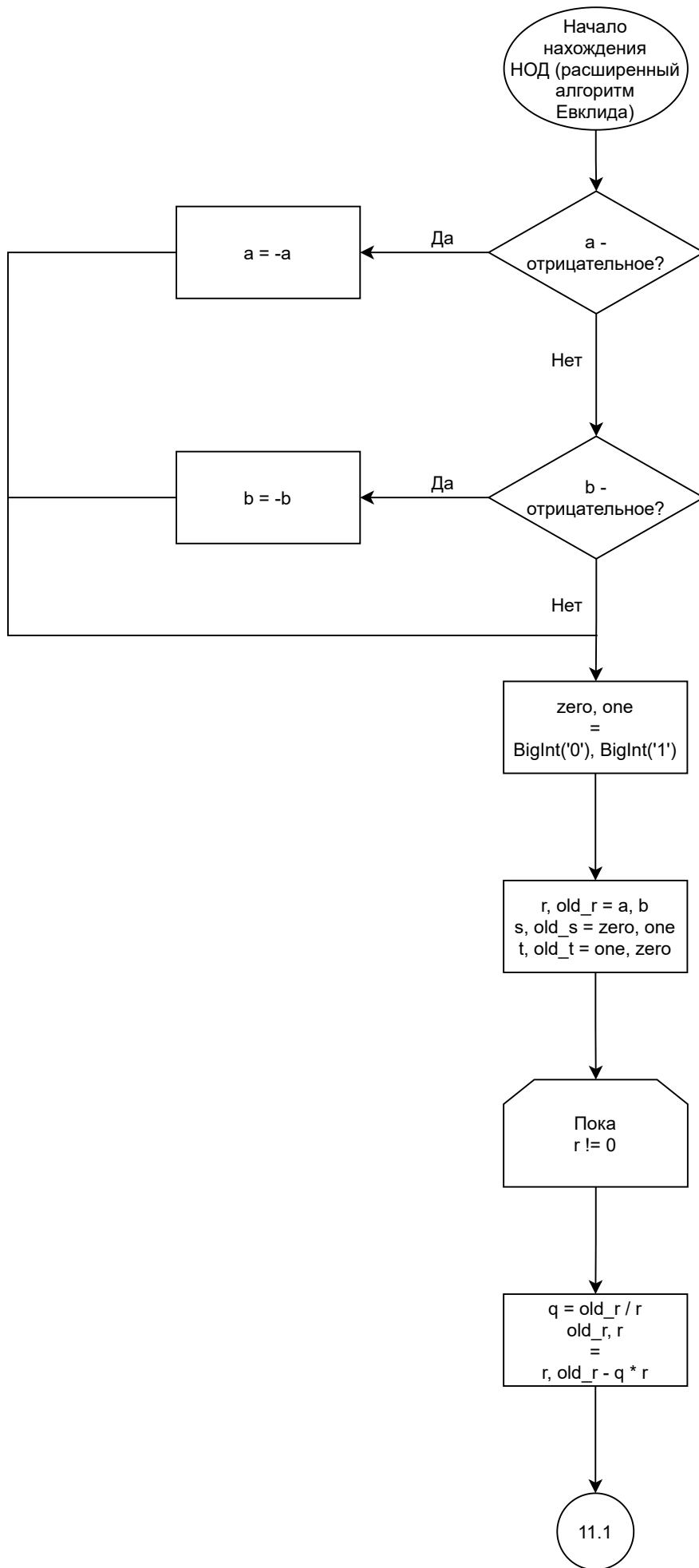


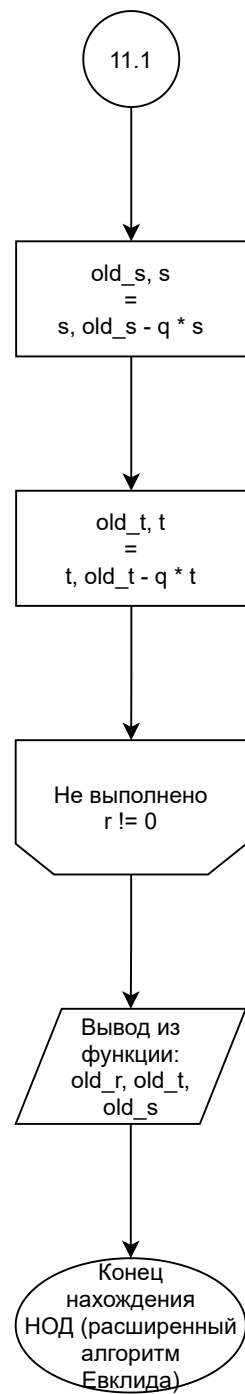


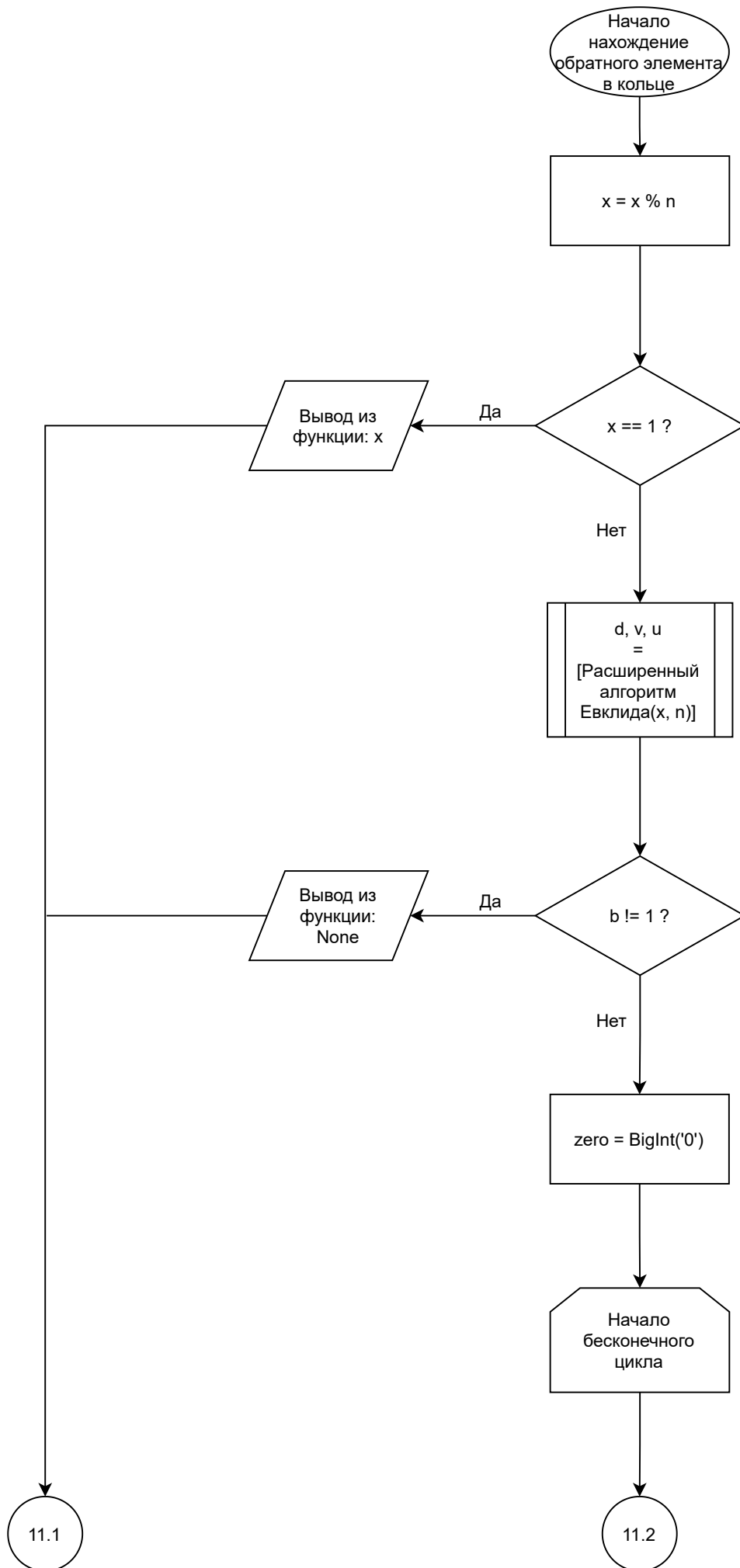


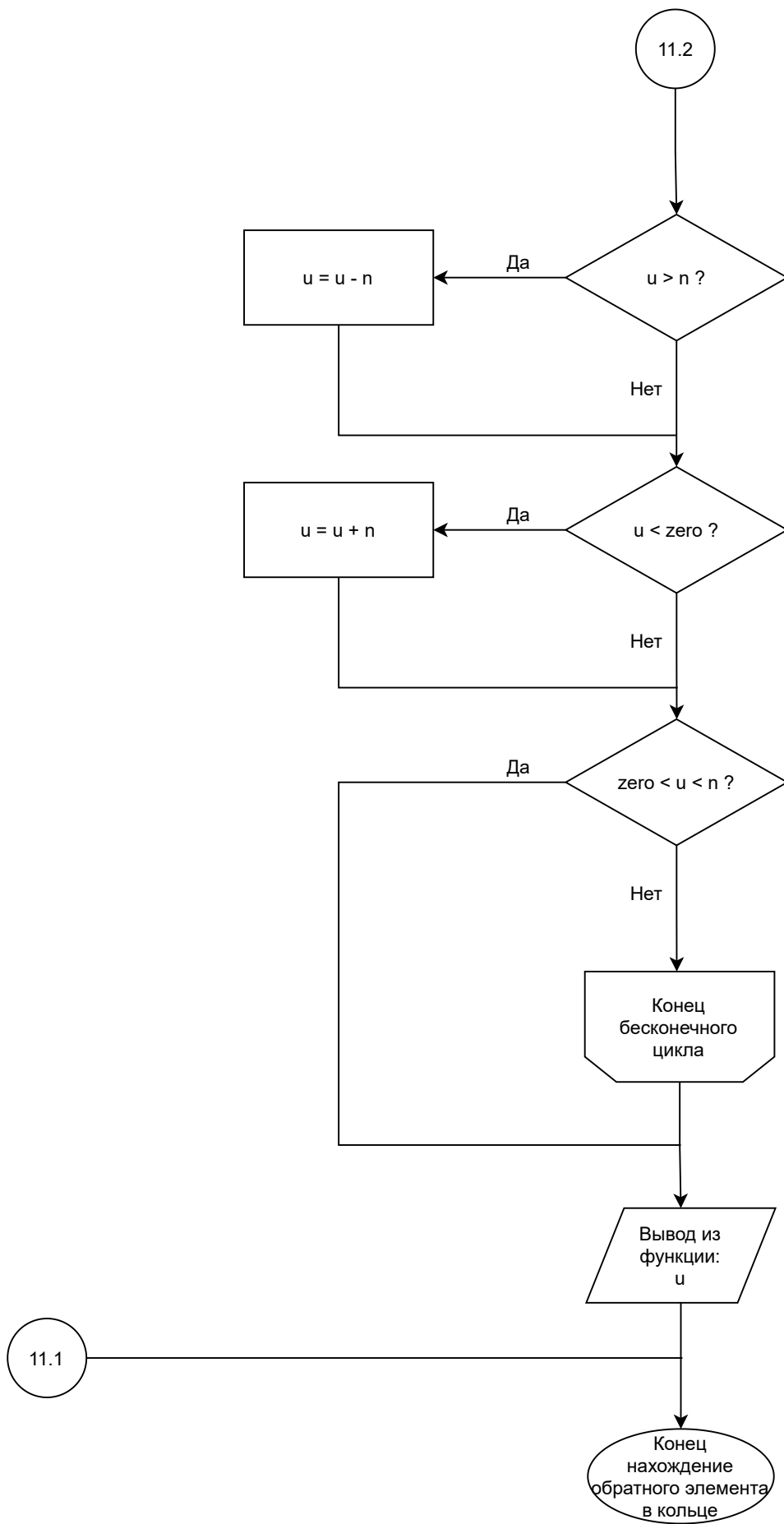












7 Исходный код программы

```
1  # -*- coding: utf-8 -*-
2
3  import time
4  from sys import setrecursionlimit
5
6  setrecursionlimit(1500) # Максимальный стек рекурсии
7
8
9  class BigInt(object):
10     is_neg = False # Флаг отрицательности числа
11     value = ''# Число в виде строки
12
13     def __init__(self, x=0):
14         self.value = '0'
15         # Если в конструктор передано целое число
16         if isinstance(x, int):
17             self.is_neg = x < 0
18             self.value = str(x if x >= 0 else -x)
19         # Если в конструктор передана строка
20         elif isinstance(x, str):
21             # Если вдруг пришла пустая строка, пропускаем, оставляем value =
22             # 0
23             if len(x):
24                 self.is_neg = x[0] == '-'
25                 # Значением будет все, после минуса, если он был. И убираем
26                 # ведущие нули, если они были
27                 self.value = x[self.is_neg:].lstrip('0')
28                 # Проверяем, является ли строка числом, если нет, value = 0
29                 if not self.value.isdigit():
30                     self.value = '0'
31                 if self.value == '0':
32                     self.is_neg = False
33             # Если в конструктор передан экземпляр того же класса, копируем его
34             # содержимое
35             elif isinstance(x, BigInt):
36                 self.value = x.value
37                 self.is_neg = x.is_neg
38
39     # Является ли число четным
40     def is_even(self):
41         return not (int(self.value[-1]) & 1)
```

```

39
40     # Перегрузка числа по модулю
41     def __abs__(self):
42         return BigInt(self.value)
43
44     def bipow(self, n):
45         # Любое число в степени 0 = 1
46         if n < 0:
47             return None
48         if not n:
49             return BigInt(1)
50         b = bin(n)[2:]
51         res = self
52         for i in range(1, len(b)):
53             res = res * res
54             if b[i] == '1':
55                 res = res * self
56         return res
57
58     def birt(self, n):
59         # Корень извлекать можем только из положительного числа
60         if (n < 0) or self.is_neg:
61             return None
62         if n == 1:
63             return self
64         length = (len(self.value) + 1) // 2
65         index = 0
66         v = [0] * length
67         while index < length:
68             v[index] = 9
69             while BigInt(''.join(str(x) for x in v)).bipow(n) > self and v[
              index]:
70                 v[index] -= 1
71             index += 1
72         v = ''.join(str(x) for x in v).lstrip('0')
73         return BigInt('-' + v) if self.is_neg else BigInt(v)
74
75     # Перегрузка перевода в bool
76     def __bool__(self):
77         return self.value != '0'
78

```

```

79     # Перегрузка x < y
80     def __lt__(self, other):
81         if isinstance(other, int):
82             other = BigInt(other)
83         self_len = len(self.value) # Запоминаем длину первого числа
84         self_other = len(other.value) # Запоминаем длину второго числа
85         # Если знаки одинаковые, то проверяем значения
86         if self.is_neg == other.is_neg:
87             # Если длины не равны
88             if self_len != self_other:
89                 # Меньше число с меньшей длиной для положительных и с
90                 # большей длиной для отрицательных
91                 return (self_len < self_other) ^ self.is_neg
92             i = 0
93             # Ищем разряд, в котором значения отличаются
94             while (i < self_len and self.value[i] == other.value[i]):
95                 i += 1
96             # Если разряд найден, то меньше число с меньшей цифрой для
97             # положительных и с большей цифрой для отрицательных, иначе
98             # числа равны
99             return (i < self_len) and ((self.value[i] < other.value[i]) ^
100                                     self.is_neg)
101         return self.is_neg # Знаки разные, если число отрицательное, то оно
102         # меньше, если положительное, то больше
103
104     # Перегрузка x <= y
105     def __le__(self, other):
106         return self < other or self == other
107
108     # Перегрузка x == y
109     def __eq__(self, other):
110         if isinstance(other, int):
111             other = BigInt(other)
112         return (self.value == other.value) and (self.is_neg == other.is_neg)
113
114     # Перегрузка x != y
115     def __ne__(self, other):
116         return not self == other
117
118     # Перегрузка x > y
119     def __gt__(self, other):
120         return not (self < other or self == other)
121

```

```

117     # Перегрузка x >= y
118     def __ge__(self, other):
119         return self > other or self == other
120
121     # Унарный плюс (просто копируем значение числа)
122     def __pos__(self):
123         return self
124
125     # Унарный минус
126     def __neg__(self):
127         if self == 0:
128             return self
129         return BigInt(self.value if self.is_neg else '-' + self.value)
130
131     # Число в бинарный вид (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
132     def to_bin(self):
133         return bin(int(self.value))
134
135     # Битовый сдвиг вправо (x » y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
136     def __rshift__(self, n):
137         if n < 0:
138             raise ValueError
139         self_bin = self.to_bin()
140         if n >= len(self_bin) - 2 or (self == 0):
141             return BigInt(0)
142         return BigInt(int(self_bin[:len(self_bin) - n], 2))
143
144     # Битовый сдвиг влево (x « y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
145     def __lshift__(self, n):
146         if n < 0:
147             raise ValueError
148         if self == 0:
149             return BigInt(0)
150         self_bin = self.to_bin()
151         return BigInt(int(self_bin + ('0' * n), 2))
152
153     # Побитовое И (x & y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
154     def __and__(self, other):
155         if isinstance(other, int):
156             other = BigInt(other)
157         self_bin = self.to_bin()[2:]

```

```

158     other_bin = other.to_bin()[2:]
159     self_len = len(self_bin)
160     other_len = len(other_bin)
161     if self_len > other_len:
162         other_bin = other_bin.zfill(self_len)
163     elif self_len < other_len:
164         self_bin = self_bin.zfill(other_len)
165     res = int('0b' + ''.join(['1' if (x, y) == ('1', '1') else '0' for x,
166                               y in zip(self_bin, other_bin)]), 2)
167     return BigInt(res)
168
169 # Побитовое ИЛИ (x | y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
170 def __or__(self, other):
171     if isinstance(other, int):
172         other = BigInt(other)
173     self_bin = self.to_bin()[2:]
174     other_bin = other.to_bin()[2:]
175     self_len = len(self_bin)
176     other_len = len(other_bin)
177     if self_len > other_len:
178         other_bin = other_bin.zfill(self_len)
179     elif self_len < other_len:
180         self_bin = self_bin.zfill(other_len)
181     res = int('0b' + ''.join(['0' if (x, y) == ('0', '0') else '1' for x,
182                               y in zip(self_bin, other_bin)]), 2)
183     return BigInt(res)
184
185 # Возврат копии
186 def copy(self):
187     return BigInt(('-' if self.is_neg else '') + self.value)
188
189 # Сложение двух чисел
190 def __add__(self, other):
191     if isinstance(other, int):
192         other = BigInt(other)
193     if self == 0:
194         return other
195     if other == 0:
196         return self
197     # Если знаки одинаковые, то выполняем сложение
198     if other.is_neg == self.is_neg:

```

```

197         num2 = other.value # Запоминаем значение второго операнда
198         self_len = len(self.value) # Длина первого операнда
199         other_len = len(num2) # Длина второго операнда
200         # Длина суммы равна максимуму из двух длин + 1 из-за возможного
                переноса разряда
201         length = max(self_len, other_len)
202         res = [0] * (length + 1)
203         for i in range(length):
204             j = length - i
205             # Выполняем сложение разрядов
206             res[j] += int((num2[other_len - 1 - i] if i < other_len else '
                0')) + int((self.value[self_len - 1 - i] if i < self_len
                else '0'))
207             res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
                разряд, если он был
208             res[j] = res[j] % 10 # Оставляем только единицы от возможного
                переноса и превращаем символ в цифру
209             # Возвращаем результат, учитывая его знак
210         return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for x
                in res))
211
212         # Если одно из чисел отрицательное, а другое положительное,
                отправляем на вычитание, меняя знак
212         return (self - (-BigInt(other))) if self.is_neg else (other - (-
                BigInt(self)))
213
214     # Вычитание одного числа из другого
215     def __sub__(self, other):
216         if isinstance(other, int):
217             other = BigInt(other)
218         # Если числа равны, считать не нужно
219         if self == other:
220             return BigInt(0)
221         if self == 0:
222             return -other
223         if other == 0:
224             return self
225         # Если оба числа положительные, выполняем вычитание
226         if not self.is_neg and not other.is_neg:
227             self_len = len(self.value) # Запоминаем длину первого числа
228             self_other = len(other.value) # Запоминаем длину второго числа
229             length = max(self_len, self_other) - 1 # Длина результата не
                превысит максимума длин чисел
230             is_neg_res = other > self # Определяем знак результата
231             # Массивы аргументов

```



```

232         new_length = length + 1
233         a = [0] * new_length
234         b = [0] * new_length
235         res = [0] * new_length
236         sign = 2 * is_neg_res - 1 # Получаем числовое значение знака
                                     результата
237         for i in range(length):
238             a[i] += int(self.value[self_len - 1 - i]) if i < self_len
                                     else 0 # Формируем
                                     разряды
239             b[i] += int(other.value[self_other - 1 - i]) if i <
                                     self_other else 0 # Из строк
                                     аргументов
240             b[i + 1] = -is_neg_res # В зависимости от знака занимаем или
                                     не занимаем
241             a[i + 1] = is_neg_res - 1 # 10 у следующего разряда
242             res[length - i] += 10 + sign * (b[i] - a[i])
243             res[length - 1 - i] = res[length - i] // 10
244             res[length - i] = res[length - i] % 10
245         # Выполняем операцию с последним разрядом
246         a[length] += (length < self_len) * int(self.value[0])
247         b[length] += (length < self_other) * int(other.value[0])
248         # Записываем в строку последний разряд
249         res[0] += sign * (b[length] - a[length])
250         # Возвращаем результат, учитывая его знак
251         return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                                     in res))
252     return -BigInt(other) - (-BigInt(self)) if self.is_neg and other.
        is_neg else self + -BigInt(other)
253
254 # Умножение двух чисел
255 def __mul__(self, other):
256     if isinstance(other, int):
257         other = BigInt(other)
258     # Если один из множителей равен нулю, то результат равен нулю
259     if self.value == '0' or other.value == '0':
260         return BigInt(0)
261     self_len = len(self.value) # Запоминаем длину первого числа
262     other_len = len(other.value) # Запоминаем длину второго числа
263     length = self_len + other_len # Результат влезет в сумму длин + 1
                                     из-за возможного переноса
264     # Флаг отрицательности результата - отрицательный, если числа разных
                                     знаков
265     is_neg_res = self.is_neg ^ other.is_neg
266     if length < 10: # Число небольшое, можно по нормальному

```

```

267         res = int(self.value) * int(other.value)
268         return BigInt(-res if is_neg_res else res)
269     else: # Умножаем в столбик
270         # Массивы аргументов
271         new_length = length + 1
272         a = [0] * new_length
273         b = [0] * new_length
274         res = [0] * new_length
275         # Заполняем массивы инверсной записью чисел (с ведущими нулями)
276         for i in range(new_length):
277             a[i] = int(self.value[self_len - 1 - i]) if i < self_len else
                0
278             b[i] = int(other.value[other_len - 1 - i]) if i < other_len
                else 0
279         # Выполняем умножение "в столбик"
280         for i in range(self_len):
281             for j in range(other_len):
282                 res[length - (i + j)] += a[i] * b[j]
283                 res[length - (i + j + 1)] += res[length - (i + j)] // 10
284                 res[length - (i + j)] %= 10
285         # Возвращаем результат, учитывая его знак
286         return BigInt(('-' if is_neg_res else '') + ''.join(str(x) for x
                in res))
287
288     # Деление одного числа на другое
289     def __truediv__(self, other):
290         if isinstance(other, int):
291             other = BigInt(other)
292         value1 = self.value # Запоминаем значение первого числа
293         value2 = other.value # Запоминаем значение второго числа
294         if value2 == '0':
295             raise ZeroDivisionError # Нельзя делить на ноль
296         if value1 == '0':
297             return BigInt(0) # А вот ноль делить можно на всё, кроме нуля, но
                СМЫСЛ
298         if value2 == '1':
299             return -BigInt(self) if other.is_neg else BigInt(self) # Делить
                на 1 можно, но смысл?
300         zeroes = 0
301         while value2[len(value2) - 1 - zeroes] == '0':
302             zeroes += 1
303         if zeroes >= len(value1):
304             return BigInt(0)

```

```

305     # если у нас 13698 / 1000, то мы можем делить 13 / 1
306     if zeroes:
307         value1 = value1[:len(value1) - zeroes]
308         value2 = value2[:len(value2) - zeroes]
309     is_neg_res = self.is_neg ^ other.is_neg # Считаем знак числа
310     tmp = BigInt(value2)
311     divider_length = len(value2) # Запоминаем длину делителя
312     # Если длина больше 8, то обнуляем делитель, иначе переводим строку
        в long
313     # Можно не обнулять, но мы думаем, что Python не умеет в большие
        числа
314     divider_v = 0 if divider_length > 8 else int(value2)
315     length = len(value1) # Получаем длину делимого
316     index = 0 # Стартуем с нулевого индекса
317     div = ''# Строка результата деления
318     v = ''# Строка подчисла (которое делится на делитель в столбик)
319     index = len(value2)
320     v = value1[:index]
321     mod = None
322     while True:
323         count = 0 # Результат деления подчисла на делитель
324         # Если можем разделить, то делим
325         if BigInt(v) >= tmp:
326             # Если не входит в long, то делим с помощью вычитания
327             if divider_length > 8:
328                 mod = BigInt(v)
329                 while mod >= tmp:
330                     mod = (mod - tmp).copy()
331                     count += 1
332                 v = mod.value
333             else:
334                 mod = int(v)
335                 count = mod // divider_v
336                 v = str(mod % divider_v)
337         # Если не делили, то добавили ноль к результату, иначе добавили
        результат деления
338         div = div + (str(count) if count else '0')
339         if index <= length:
340             try: # Тот самый ноль, лучше не спрашивать
341                 v = v + value1[index]
342             except IndexError:
343                 v = v + '0'
344         index += 1 # Формируем новое значение для подчисла

```

```

345         if not (index <= length):
346             break
347         # Возвращаем результат учитывая знак и возможное равенство нулю
348         return BigInt('-' + div if is_neg_res and div != '0' else div)
349
350     # Обработка для выходных данных
351     def __str__(self):
352         return str('-' if self.is_neg else '') + self.value
353
354     # Остаток от деления
355     def __mod__(self, other):
356         if isinstance(other, int):
357             other = BigInt(other)
358         if other.value == '0':
359             return None
360         if self.value == '0' or other.value == "1":
361             return BigInt(0)
362         # Если числа меньше 9, можно посчитать по нормальному
363         if len(self.value) < 9 and len(other.value) < 9:
364             return BigInt(int(str('-' if self.is_neg else '') + self.value) %
365                             int(str('-' if other.is_neg else '') + other.value))
366         tmp = BigInt(other.value)
367         divider_length = len(other.value) # запоминаем длину делителя
368         # Если длина больше 8, то обнуляем long'овый делитель, иначе
369         # переводим строку в long
370         divider_v = 0 if divider_length > 8 else int(other.value)
371         length = len(self.value)
372         index = 0
373         mod2 = self.copy()
374         v = ''
375         mod = None
376         while BigInt(v) < tmp and index < length:
377             v = v + self.value[index]
378             index += 1
379         while True:
380             if BigInt(v) >= tmp:
381                 if divider_v:
382                     v = str(int(v) % divider_v)
383                 else:
384                     mod = BigInt(v)
385                     while mod >= tmp:
386                         mod = (mod - tmp).copy()

```

```

385         v = mod.value
386     if index <= length:
387         mod2 = v
388         try:
389             v = v + self.value[index]
390         except IndexError:
391             break
392         index += 1
393     if not (index <= length):
394         break
395     if isinstance(mod2, BigInt):
396         if mod2.value == '0':
397             return BigInt(0)
398     res = -BigInt(mod2) if self.is_neg else BigInt(mod2)
399     if self.is_neg ^ other.is_neg and res != 0:
400         return other + res
401     return res
402
403     # Сложение в кольце вычетов
404     @staticmethod
405     def ring_add(a, b, m):
406         if isinstance(a, int):
407             a = BigInt(a)
408         if isinstance(b, int):
409             b = BigInt(b)
410         if isinstance(m, int):
411             m = BigInt(m)
412         if m < 1:
413             return None
414         return abs(a + b) % m
415
416     # Вычитание в кольце вычетов
417     @staticmethod
418     def ring_sub(a, b, m):
419         if isinstance(a, int):
420             a = BigInt(a)
421         if isinstance(b, int):
422             b = BigInt(b)
423         if isinstance(m, int):
424             m = BigInt(m)
425         if m < 1:

```

```

426         return None
427     return abs(a - b) % m
428
429     # Умножение в кольце вычетов
430     @staticmethod
431     def ring_mul(a, b, m):
432         if isinstance(a, int):
433             a = BigInt(a)
434         if isinstance(b, int):
435             b = BigInt(b)
436         if isinstance(m, int):
437             m = BigInt(m)
438         if m < 1:
439             return None
440         return abs(a * b) % m
441
442     # Расширенный алгоритм Евклида
443     @staticmethod
444     def Evclid_GCD(a, b):
445         if isinstance(a, int):
446             a = BigInt(a)
447         if isinstance(b, int):
448             b = BigInt(b)
449         if a < 0:
450             a = -a
451         if b < 0:
452             b = -b
453         zero, one = BigInt('0'), BigInt('1')
454         r, old_r = a, b
455         s, old_s = zero, one
456         t, old_t = one, zero
457         while r != 0:
458             q = old_r / r
459             old_r, r = r, old_r - q * r
460             old_s, s = s, old_s - q * s
461             old_t, t = t, old_t - q * t
462         return old_r, old_t, old_s
463
464     # Нахождение обратного элемента в кольце
465     @staticmethod
466     def ring_inv_el(x, n):

```

```

467         x = x % n
468     if x == 1:
469         return x
470     d, v, u = BigInt.Evclid_GCD(x, n)
471     if d != 1:
472         return None
473     zero = BigInt('0')
474     while True:
475         if u > n:
476             u = u - n
477         if u < zero:
478             u = u + n
479         if zero < u < n:
480             break
481     return u
482
483     # Нахождение степени в кольце
484     @staticmethod
485     def ring_pow(x, m, n):
486         if isinstance(x, int):
487             x = BigInt(x)
488         if isinstance(m, int):
489             m = BigInt(m)
490         if isinstance(n, int):
491             n = BigInt(n)
492         if n < 1:
493             return None
494         if m == 0:
495             return BigInt(1)
496         b = m.to_bin()[2:]
497         z = x % n
498         for i in range(1, len(b)):
499             z = (z * z) % n
500             if b[i] == '1':
501                 z = (z * x) % n
502         return z
503
504
505     def GCD(a, b):
506         if a < 0:
507             a = -a

```

```

508     if b < 0:
509         b = -b
510     while b:
511         a, b = b, a % b
512     return a
513
514
515 def binary_GCD(num1, num2):
516     if num1 < 0:
517         num1 = -num1
518     if num2 < 0:
519         num2 = -num2
520     shift = 0
521     # Если одно из чисел равно нулю, делитель - другое число
522     if num1 == 0:
523         return num2
524     if num2 == 0:
525         return num1
526     # Если num1 = 1010, а num2 = 0100, то num1 | num2 = 1110
527     # 1110 & 0001 == 0, тогда происходит сдвиг, который фиксируется в shift
528     while (num1 | num2) & 1 == 0:
529         shift += 1
530         num1 = num1 >> 1
531         num2 = num2 >> 1
532     # Если True, значит num1 - четное, иначе - нечетное
533     while num1 & 1 == 0:
534         # если нечетное, сдвигаем на один бит
535         num1 = num1 >> 1
536     while num2 != 0:
537         # пока число нечётное, сдвигаем на один бит
538         while num2 & 1 == 0:
539             num2 = num2 >> 1
540         # если первое число больше второго
541         if num1 > num2:
542             # меняем их местами
543             num1, num2 = num2, num1
544         # теперь первое число меньше второго, вычитаем
545         num2 = num2 - num1
546     # возвращаем число, перед этим сдвинув его биты на shift
547     return num1 << shift
548

```



```

549
550 if __name__ == '__main__':
551     while True:
552         menu_text = '\n'.join([
553             'Выберите действие:',
554             '1)  $x + y$ ',
555             '2)  $x - y$ ',
556             '3)  $x * y$ ',
557             '4)  $x / y$ ',
558             '5) Возведение чисел в степень  $y$ ',
559             '6) Извлечение корня из чисел степени  $y$ ',
560             '7) Нахождение НОД  $x$  и  $y$  с помощью алгоритма Евклида',
561             '8) Нахождение НОД  $x$  и  $y$  с помощью бинарного алгоритма',
562             '9) Сложение в кольце вычетов',
563             '10) Вычитание в кольце вычетов',
564             '11) Умножение в кольце вычетов',
565             '12) Найти элемент, обратный к элементу, взаимно простым с модулем кольца  $a$ ',
566             '13) Возвести в натуральную степень элемент кольца вычетов',
567             'q) Выход'
568         ])
569     print(menu_text)
570     choice = input()
571     if choice == '1':
572         x = BigInt(input('Введите первое число(x): '))
573         y = BigInt(input('Введите второе число(y): '))
574         t = time.time()
575         res = x + y
576         print(f'\nВремя расчета: {time.time() - t} сек.')
577         print('x + y =', res, '\n\n')
578     elif choice == '2':
579         x = BigInt(input('Введите первое число(x): '))
580         y = BigInt(input('Введите второе число(y): '))
581         t = time.time()
582         res = x - y
583         print(f'\nВремя расчета: {time.time() - t} сек.')
584         print('x - y =', res, '\n\n')
585     elif choice == '3':
586         x = BigInt(input('Введите первое число(x): '))
587         y = BigInt(input('Введите второе число(y): '))
588         t = time.time()

```

```

589         res = x * y
590         print(f'\nВремя расчёта: {time.time() - t} сек.')
591         print('x * y =', res, '\n\n')
592     elif choice == '4':
593         x = BigInt(input('Введите первое число(x): '))
594         y = BigInt(input('Введите второе число(y): '))
595         t = time.time()
596         res = x / y
597         print(f'\nВремя расчёта: {time.time() - t} сек.')
598         print('x / y =', res, '\n\n')
599     elif choice == '5':
600         x = BigInt(input('Введите первое число(x): '))
601         y = int(input('Введите второе число(y): '))
602         t = time.time()
603         res = x.bipow(y)
604         print(f'\nВремя расчёта: {time.time() - t} сек.')
605         print('x**y =', res, '\n\n')
606     elif choice == '6':
607         x = BigInt(input('Введите первое число(x): '))
608         y = int(input('Введите второе число(y): '))
609         t = time.time()
610         res = x.birt(y)
611         print(f'\nВремя расчёта: {time.time() - t} сек.')
612         print('x**(1/y) =', res, '\n\n')
613     elif choice == '7':
614         x = BigInt(input('Введите первое число(x): '))
615         y = BigInt(input('Введите второе число(y): '))
616         t = time.time()
617         res = GCD(x, y)
618         print(f'\nВремя расчёта: {time.time() - t} сек.')
619         print('GCD(x, y) =', res, '\n\n')
620     elif choice == '8':
621         x = BigInt(input('Введите первое число(x): '))
622         y = BigInt(input('Введите второе число(y): '))
623         t = time.time()
624         res = binary_GCD(x, y)
625         print(f'\nВремя расчёта: {time.time() - t} сек.')
626         print('binary_GCD(x, y) =', res, '\n\n')
627     elif choice == '9':
628         x = BigInt(input('Введите первое число(x): '))
629         y = BigInt(input('Введите второе число(y): '))

```

```

630         m = BigInt(input('Введите модуль(m): '))
631         t = time.time()
632         res = BigInt.ring_add(x, y, m)
633         print(f'\nВремя расчета: {time.time() - t} сек.')
634         print('ring_add(x, y, m) =', res, '\n\n')
635     elif choice == '10':
636         x = BigInt(input('Введите первоечисло(x): '))
637         y = BigInt(input('Введите второечисло(y): '))
638         m = BigInt(input('Введите модуль(m): '))
639         t = time.time()
640         res = BigInt.ring_sub(x, y, m)
641         print(f'\nВремя расчета: {time.time() - t} сек.')
642         print('ring_sub(x, y, m) =', res, '\n\n')
643     elif choice == '11':
644         x = BigInt(input('Введите первоечисло(x): '))
645         y = BigInt(input('Введите второечисло(y): '))
646         m = BigInt(input('Введите модуль(m): '))
647         t = time.time()
648         res = BigInt.ring_mul(x, y, m)
649         print(f'\nВремя расчета: {time.time() - t} сек.')
650         print('ring_mul(x, y, m) =', res, '\n\n')
651     elif choice == '12':
652         x = BigInt(input('Введите первоечисло(x): '))
653         y = BigInt(input('Введите второечисло(y): '))
654         t = time.time()
655         res = BigInt.ring_inv_el(x, y)
656         print(f'\nВремя расчета: {time.time() - t} сек.')
657         print('ring_inv_el(x, y) =', res, '\n\n')
658     elif choice == '13':
659         x = BigInt(input('Введите первоечисло(x): '))
660         y = BigInt(input('Введите второечисло(y): '))
661         m = BigInt(input('Введите модуль(m): '))
662         t = time.time()
663         res = BigInt.ring_pow(x, y, m)
664         print(f'\nВремя расчета: {time.time() - t} сек.')
665         print('ring_pow(x, y, m) =', res, '\n\n')
666     else:
667         exit()

```

Список литературы

1. *Акритас А.* Основы компьютерной алгебры с приложениями. — М. : МИР, 1994.
2. *Майрова С. П., Завгородний М. Г.* Программирование. Криптографические алгоритмы: учебное пособие. — Воронеж : Издательский дом ВГУ, 2018.
3. *Саммерфилд М.* Программирование на Python 3. Подробное руководство. — Символ-Плюс, 2009.