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

Обучающийся _____ А.А. Уткин
подпись

Воронеж 2021

Содержание

1 Постановка задачи

Составьте блок-схему и напишите соответствующую ей программу, позволяющую вычислять наибольший общий делитель двух больших натуральных чисел.

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», рядом с которым объявлены следующие функции:

- Реализации алгоритма Евклида для поиска НОД двух длинных целых чисел.
  Алгоритм был реализован для сравнения с более сложным алгоритмом того же назначения.

- Реализации бинарного алгоритма для поиска НОД двух длинных целых чисел.
  Данный алгоритм является более оптимальным, по сравнению с алгоритмом Евклида. Но в рамках использования языка Python он оказался более медленным, за счет частых преобразований строковых переменных в логические, для последующих операций битовых сдвигов.

- Реализации алгоритма «LSBGCD» для поиска НОД двух длинны х целых чисел, а так же их коэффициенты u, v.
  Данный алгоритм был реализован с целью получения коэффициентов u и v.

Функции алгоритма Евклида для поиска НОД и инарного алгоритма для поиска НОД не являются методами класс «BigInt», так как были реализованны таким образом, что бы они могли работать как с экземплярами класса «BigInt», так и с обычными переменным типа «int».

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», а так же отдельных функций.

Пусть даны два целых длинных числа a и b, сохраненных в экземпляр класса «BigInt». А так же, создадим экземпляр класса «BigInt» с нулевым значением.

```
1 a = BigInt('1234567890987654321')
2 b = BigInt('9876543210123456789')
3 zero = BigInt()
```

- Выполняем нахождение НОД двух длинных целых чисел с помощью алгоритма Евклида:

```
1 print(GDC(a, b))
```

Вывод: 9

- Выполняем нахождение НОД двух длинных целых чисел с бинарного алгоритма:

```
1 print(binary_GCD(a, b))
```

Вывод: 9

- Выполняем нахождение НОД двух длинных целых чисел, а так же их коэффициенты u, v с помощью алгоритма «LSBGCD»:

```
1 d, u, v = BigInt.lsbgcd(x, y)
2 print(f'd = {d}, u = {u}, v = {v}')
```

Вывод: $d = 9$, $u = 116194618262890341$, $v = -14524327161946268$

5 Вывод

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

- вычислять наибольший общий делитель двух больших натуральных чисел с помощью алгоритма Евклида;
- вычислять наибольший общий делитель двух больших натуральных чисел с помощью бинарного алгоритма;
- вычислять наибольший общий делитель двух больших натуральных чисел с помощью алгоритма «LSBGCD» а так же находить их коэффициенты u, v.

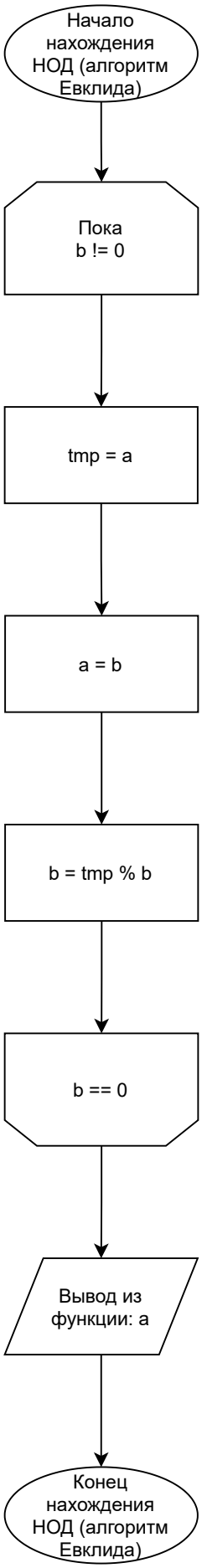
6 Блок-схема методов библиотеки

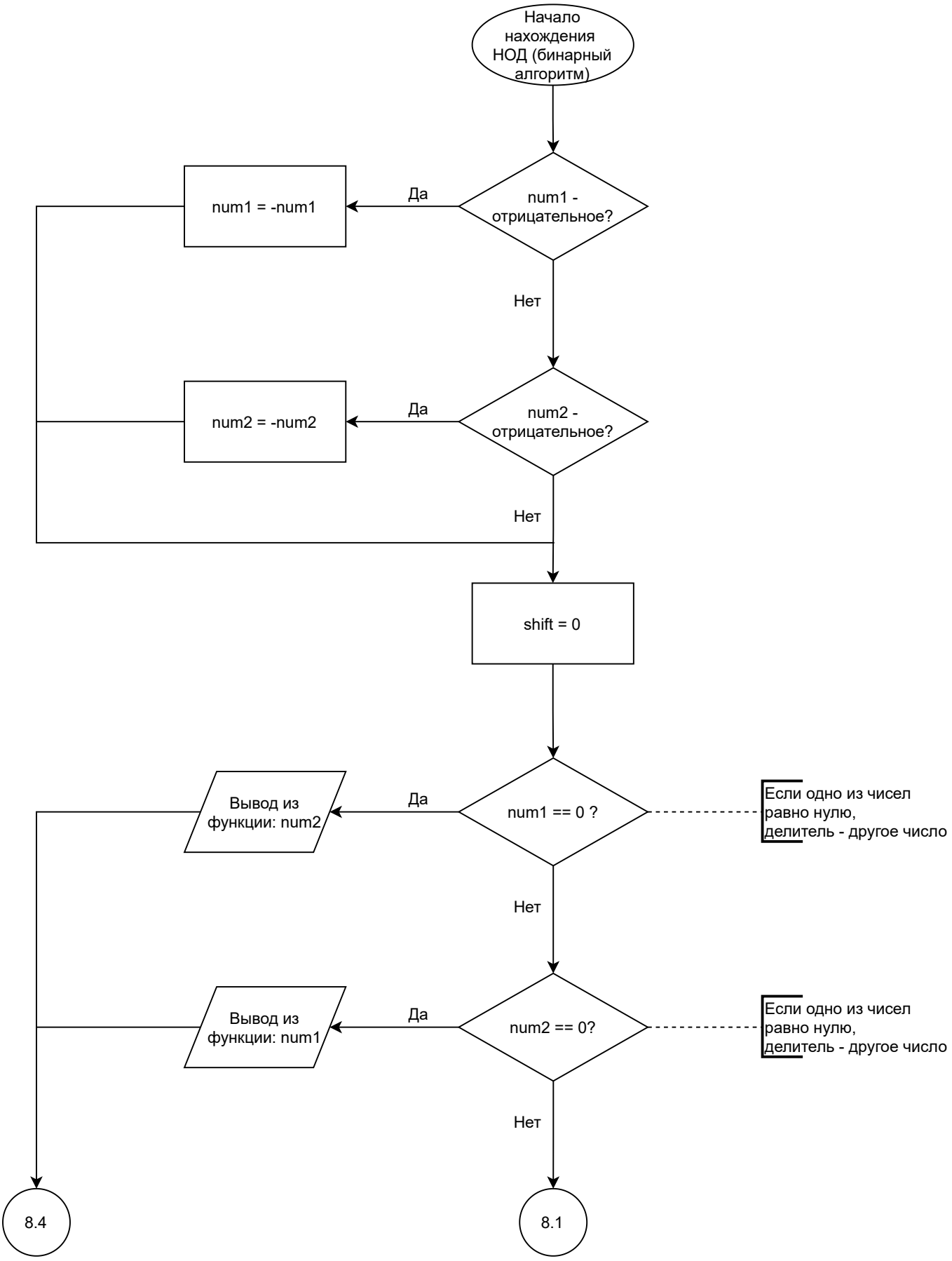
Ниже представлены блок-схемы функций в следующем порядке:

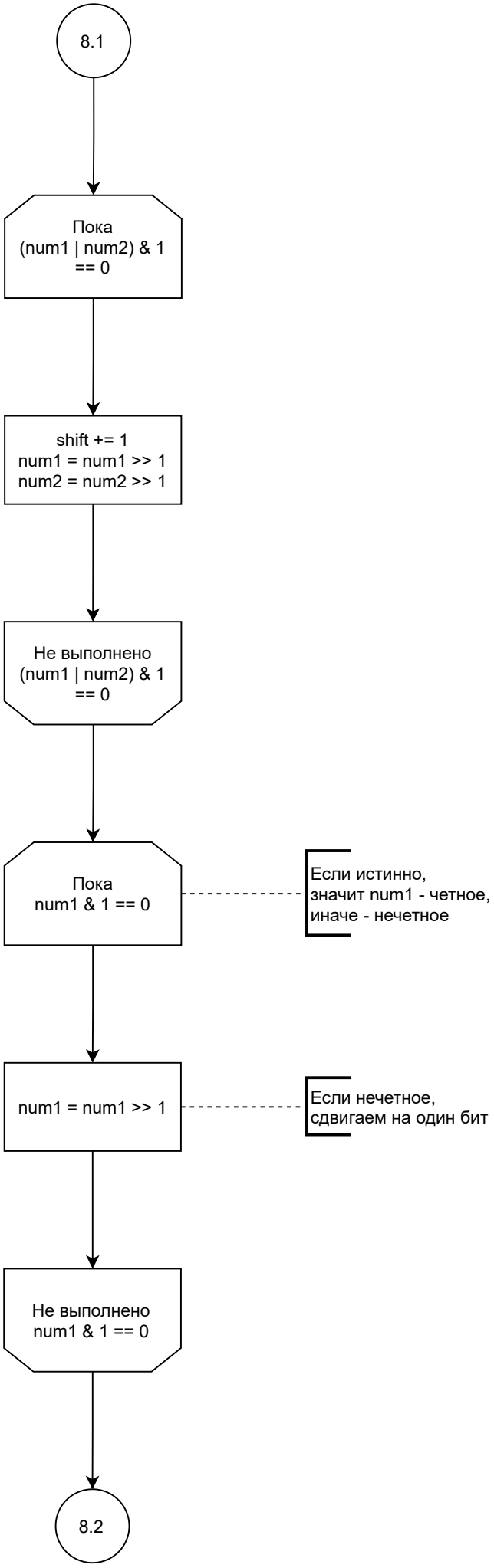
1. Функция вычисления наибольшего общего делителя двух больших натуральных чисел с помощью алгоритма Евклида;

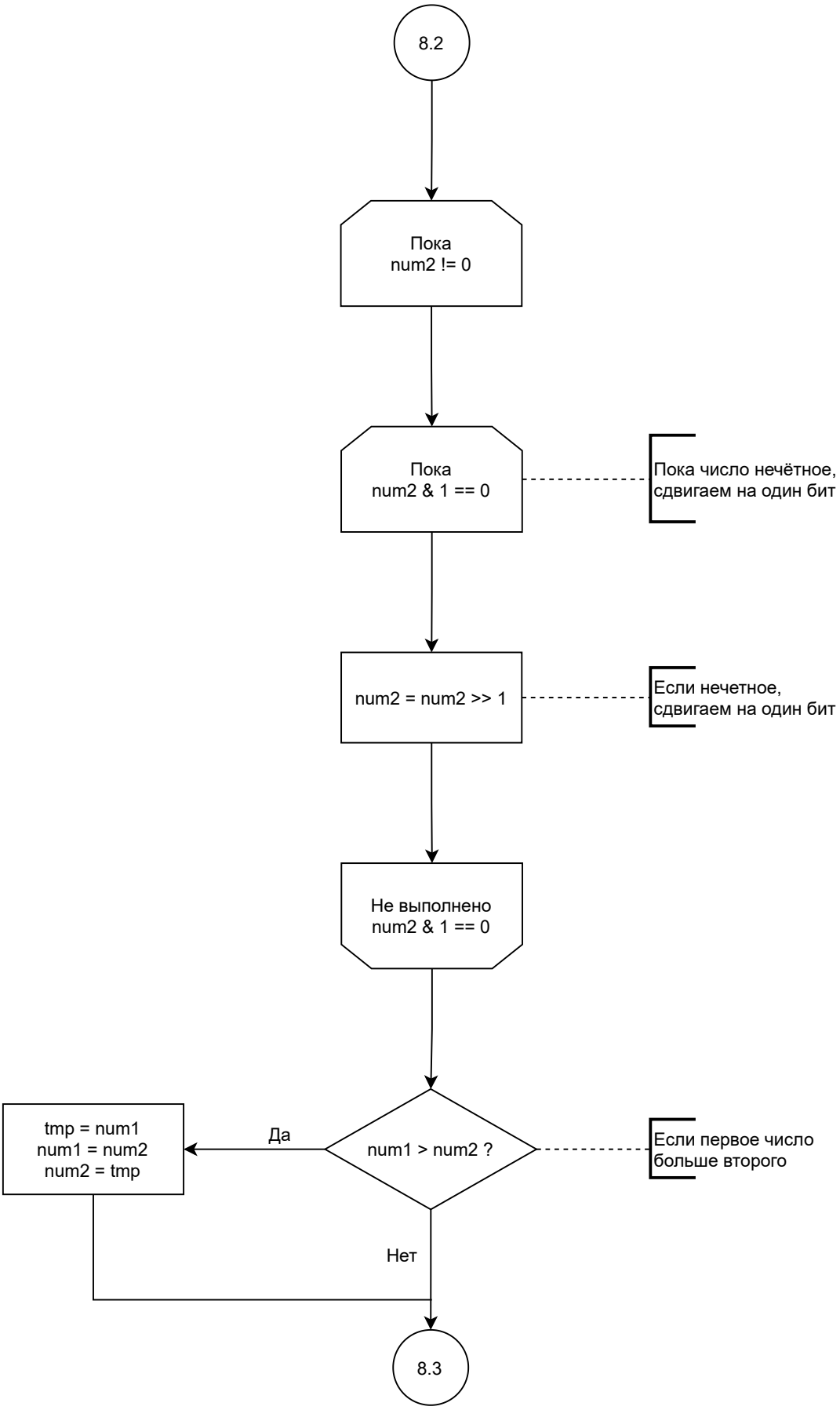
2. Функция вычисления наибольшего общего делителя двух больших натуральных чисел с помощью бинарного алгоритма;

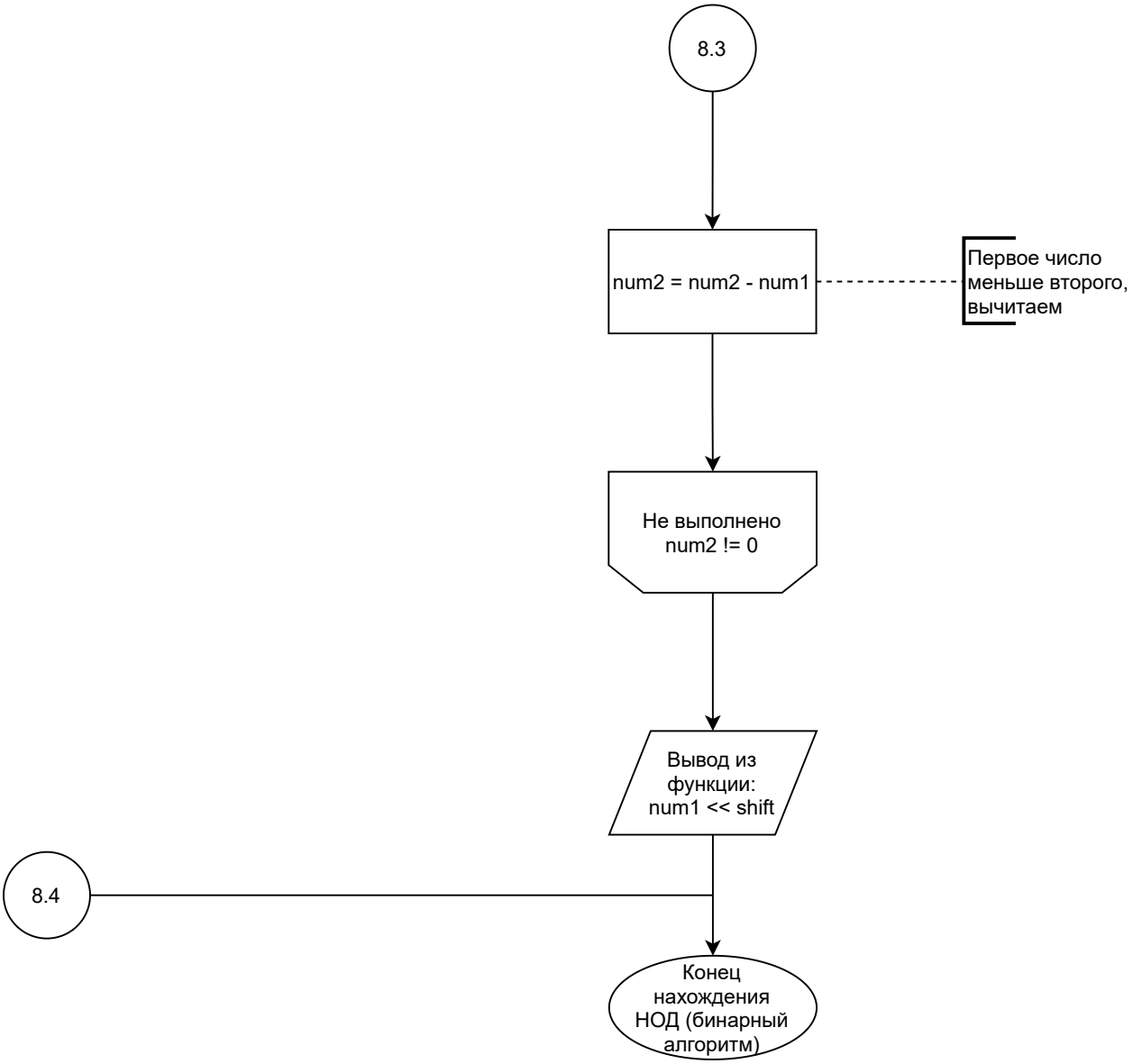
3. Метод вычисления наибольшего общего делителя двух больших натуральных чисел с помощью алгоритма «LSBGCD» а так же нахождения их коэффициентов u, v.

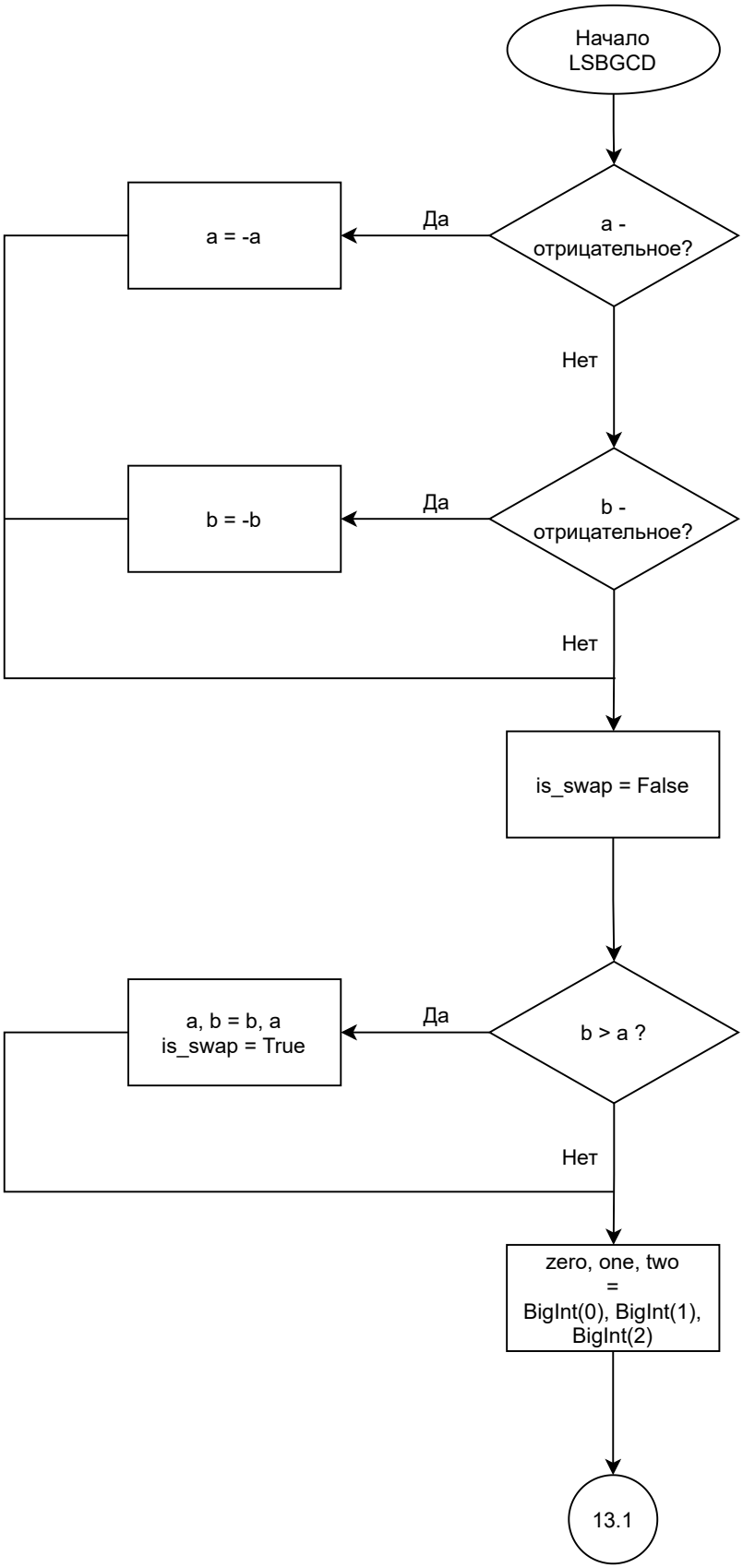

Начало нахождения НОД (алгоритм Евклида)
Пока b != 0
tmp = a
a = b
b = tmp % b
b == 0
Вывод из функции: a
Конец нахождения НОД (алгоритм Евклида)

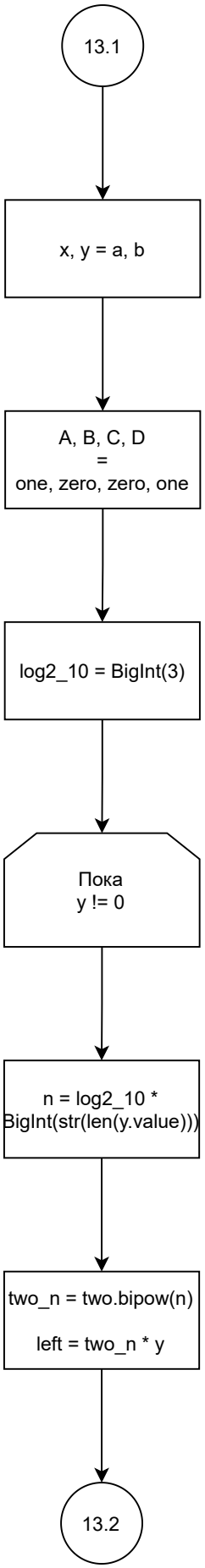

Начало нахождения НОД (бинарный алгоритм)
num1 - отрицательное?
Да
num1 = -num1
Нет
num2 - отрицательное?
Да
num2 = -num2
Нет
shift = 0
num1 == 0 ?
Да
Вывод из функции: num2
Если одно из чисел равно нулю, делитель - другое число
Нет
num2 == 0?
Да
Вывод из функции: num1
Если одно из чисел равно нулю, делитель - другое число
Нет
8.4
8.1

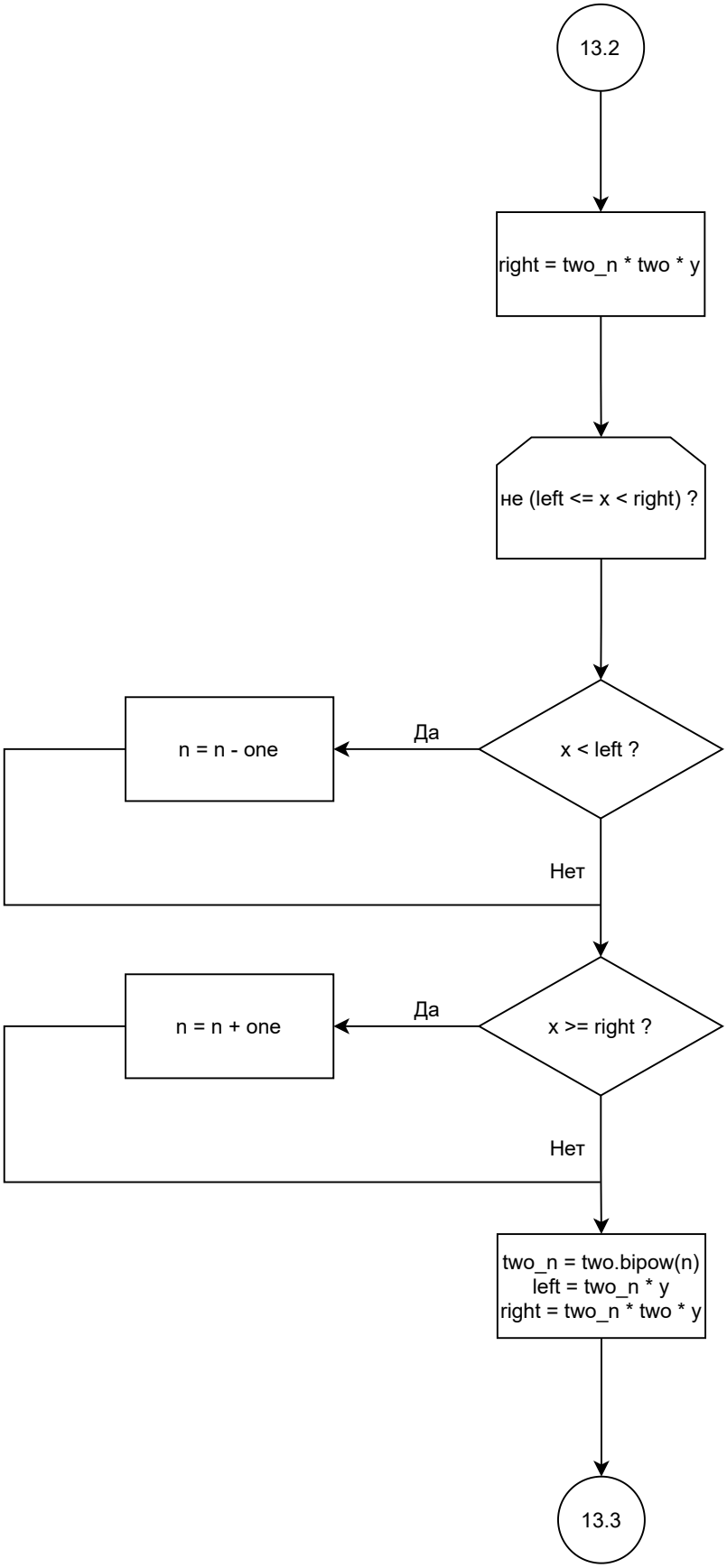

8.1
Пока
(num1 | num2) & 1
== 0
shift += 1
num1 = num1 >> 1
num2 = num2 >> 1
Не выполнено
(num1 | num2) & 1
== 0
Пока
num1 & 1 == 0
Если истинно,
значит num1 - четное,
иначе - нечетное
num1 = num1 >> 1
Если нечетное,
сдвигаем на один бит
Не выполнено
num1 & 1 == 0
8.2

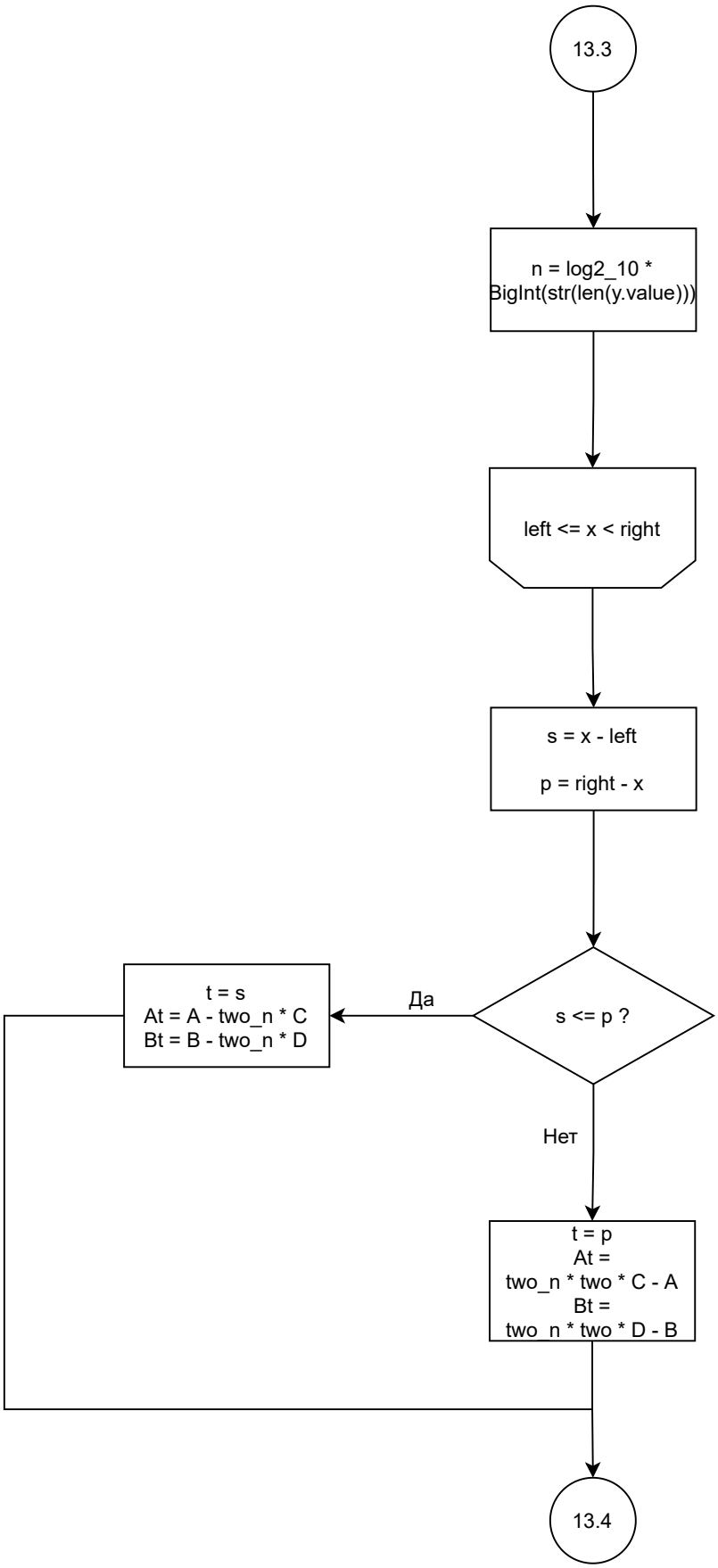

8.2
Пока
num2 != 0
Пока
num2 & 1 == 0
Пока число нечётное,
сдвигаем на один бит
num2 = num2 >> 1
Если нечетное,
сдвигаем на один бит
Не выполнено
num2 & 1 == 0
num1 > num2 ?
Если первое число
больше второго
Да
tmp = num1
num1 = num2
num2 = tmp
Нет
8.3

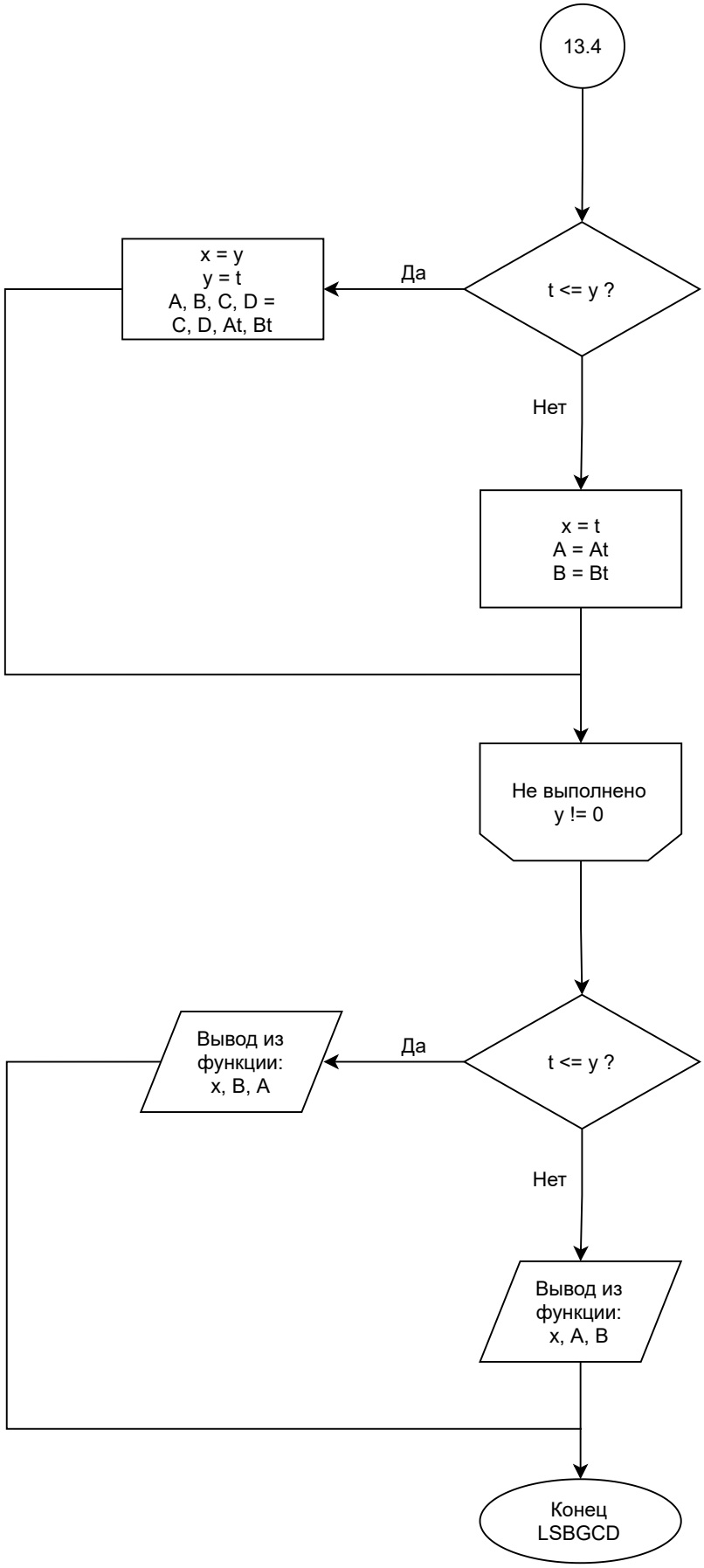

8.3
num2 = num2 - num1
Первое число меньше второго, вычитаем
Не выполнено num2 != 0
Вывод из функции: num1 << shift
8.4
Конец нахождения НОД (бинарный алгоритм)


Начало
LSBGCD
a -
отрицательное?
Да
a = -a
Нет
b -
отрицательное?
Да
b = -b
Нет
is_swap = False
b > a ?
Да
a, b = b, a
is_swap = True
Нет
zero, one, two
=
BigInt(0), BigInt(1),
BigInt(2)
13.1


13.1
x, y = a, b
A, B, C, D
=
one, zero, zero, one
log2_10 = BigInt(3)
Пока
y != 0
n = log2_10 *
BigInt(str(len(y.value)))
two_n = two.bipow(n)
left = two_n * y
13.2


13.2
right = two_n * two * y
не (left <= x < right) ?
x < left ?
Да
n = n - one
Нет
x >= right ?
Да
n = n + one
Нет
two_n = two.bipow(n)
left = two_n * y
right = two_n * two * y
13.3


13.3
n = log2_10 *
BigInt(str(len(y.value)))
left <= x < right
s = x - left
p = right - x
s <= p ?
Да
t = s
At = A - two_n * C
Bt = B - two_n * D
Нет
t = p
At =
two_n * two * C - A
Bt =
two_n * two * D - B
13.4


13.4
t <= y ?
Да
x = y
y = t
A, B, C, D =
C, D, At, Bt
Нет
x = t
A = At
B = Bt
Не выполнено
y != 0
t <= y ?
Да
Вывод из
функции:
x, B, A
Нет
Вывод из
функции:
x, A, B
Конец
LSBGCD

7 Исходный код программы

```
# -*- coding: utf-8 -*-

import time
from sys import setrecursionlimit

setrecursionlimit(1500) # Максимальный стек рекурсии


class BigInt(object):
    is_neg = False # Флаг отрицательности числа
    value = ''# Число в виде стоки

    def __init__(self, x=0):
        self.value = '0'
        # Если в конструктор передано целое число
        if isinstance(x, int):
            self.is_neg = x < 0
            self.value = str(x if x >= 0 else -x)
        # Если в конструктор передана строка
        elif isinstance(x, str):
            # Если вдруг пришла пустая строка, пропускаем, оставляем value = 0
            if len(x):
                self.is_neg = x[0] == '-'
                # Значением будет все, после минуса, если он был. И убираем ведущие нули, если они были
                self.value = x[self.is_neg:].lstrip('0')
                # Проверяем, является ли строка числом, если нет, value = 0
                if not self.value.isdigit():
                    self.value = '0'
                if self.value == '0':
                    self.is_neg = False
        # Если в конструктор передан экземпляр того же класса, копируем его содержимое
        elif isinstance(x, BigInt):
            self.value = x.value
            self.is_neg = x.is_neg

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

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

```
117         return not (self < other or self == other)
118
119     # Перегрузка x >= y
120     def __ge__(self, other):
121         return self > other or self == other
122
123     # Унарный плюс (просто копируем значение числа)
124     def __pos__(self):
125         return self
126
127     # Унарный минус
128     def __neg__(self):
129         if self == 0:
130             return self
131         return BigInt(self.value if self.is_neg else '-' + self.value)
132
133     # Число в бинарный вид (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
134     def to_bin(self):
135         return bin(int(self.value))
136
137     # Битовый сдвиг вправо (x » y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
138     def __rshift__(self, n):
139         if n < 0:
140             raise ValueError
141         self_bin = self.to_bin()
142         if n >= len(self_bin) - 2 or (self == 0):
143             return BigInt(0)
144         return BigInt(int(self_bin[:len(self_bin) - n], 2))
145
146     # Битовый сдвиг вслево (x « y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
147     def __lshift__(self, n):
148         if n < 0:
149             raise ValueError
150         if self == 0:
151             return BigInt(0)
152         self_bin = self.to_bin()
153         return BigInt(int(self_bin + ('0' * n), 2))
154
155     # Побитовое И (x & y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
156     def __and__(self, other):
157         if isinstance(other, int):
```

```
158             other = BigInt(other)
159         self_bin = self.to_bin()[2:]
160         other_bin = other.to_bin()[2:]
161         self_len = len(self_bin)
162         other_len = len(other_bin)
163         if self_len > other_len:
164             other_bin = other_bin.zfill(self_len)
165         elif self_len < other_len:
166             self_bin = self_bin.zfill(other_len)
167         res = int('0b' + ''.join(['1' if (x, y) == ('1', '1') else '0' for x,
                y in zip(self_bin, other_bin)]), 2)
168         return BigInt(res)
169
170     # Побитовое ИЛИ (x | y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
171     def __or__(self, other):
172         if isinstance(other, int):
173             other = BigInt(other)
174         self_bin = self.to_bin()[2:]
175         other_bin = other.to_bin()[2:]
176         self_len = len(self_bin)
177         other_len = len(other_bin)
178         if self_len > other_len:
179             other_bin = other_bin.zfill(self_len)
180         elif self_len < other_len:
181             self_bin = self_bin.zfill(other_len)
182         res = int('0b' + ''.join(['0' if (x, y) == ('0', '0') else '1' for x,
                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:
199             num2 = other.value # Запоминаем значение второго операнда
200             self_len = len(self.value) # Длинна первого операнда
201             other_len = len(num2) # Длинна второго операнда
202             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
                    переноса разряда
203             length = max(self_len, other_len)
204             res = [0] * (length + 1)
205             for i in range(length):
206                 j = length - i
207                 # Выполняем сложение разрядов
208                 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'))
209                 res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
                        разряд, если он был
210                 res[j] = res[j] % 10 # Оставляем только единицы от возможного
                        переноса и превращаем символ в цифру
211                 # Возвращаем результат, учитывая его знак
212             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for x
                     in res))
213         # Если одно из чисел отрицательное, а другое положительное,
                отправляем на вычитание, меняя знак
214         return (self - (-BigInt(other))) if self.is_neg else (other - (-
                BigInt(self)))
215
216     # Вычитание одного числа из другого
217     def __sub__(self, other):
218         if isinstance(other, int):
219             other = BigInt(other)
220         # Если числа равны, считать не нужно
221         if self == other:
222             return BigInt(0)
223         if self == 0:
224             return -other
225         if other == 0:
226             return self
227         # Если оба числа положительные, выполняем вычитание
228         if not self.is_neg and not other.is_neg:
229             self_len = len(self.value) # Запоминаем длину первого числа
230             self_other = len(other.value) # Запоминаем длину второго числа
231             length = max(self_len, self_other) - 1 # Длина результата не
                    превысит максимума длин чисел
```

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

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

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

```
345                 v = v + '0'
346             index += 1 # Формируем новое значение для подчисла
347         if not (index <= length):
348             break
349     # Возвращаем результат учитывая знак и возможное равенство нулю
350     return BigInt('-' + div if is_neg_res and div != '0' else div)
351
352 # Обработка для выходных данных
353 def __str__(self):
354     return str('-' if self.is_neg else '') + self.value
355
356 # Остаток от деления
357 def __mod__(self, other):
358     if isinstance(other, int):
359         other = BigInt(other)
360     if other.value == '0':
361         return None
362     if self.value == '0' or other.value == "1":
363         return BigInt(0)
364     # Если числа меньше 9, можно посчитать по нормальному
365     if len(self.value) < 9 and len(other.value) < 9:
366         return BigInt(int(str('-' if self.is_neg else '') + self.value) %
              int(str('-' if other.is_neg else '') + other.value))
367     tmp = BigInt(other.value)
368     divider_length = len(other.value) # запоминаем длину делителя
369     # Если длина больше 8, то обнуляем long'овый делитель, иначе
        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()
387                 v = mod.value
388             if index <= length:
389                 mod2 = v
390                 try:
391                     v = v + self.value[index]
392                 except IndexError:
393                     break
394                 index += 1
395             if not (index <= length):
396                 break
397         if isinstance(mod2, BigInt):
398             if mod2.value == '0':
399                 return BigInt(0)
400         res = -BigInt(mod2) if self.is_neg else BigInt(mod2)
401         if self.is_neg ^ other.is_neg and res != 0:
402             return other + res
403         return res
404 
405     # Сложение в кольце вычетов
406     @staticmethod
407     def ring_add(a, b, m):
408         if isinstance(a, int):
409             a = BigInt(a)
410         if isinstance(b, int):
411             b = BigInt(b)
412         if isinstance(m, int):
413             m = BigInt(m)
414         if m < 1:
415             return None
416         return abs(a + b) % m
417 
418     # Вычитание в кольце вычетов
419     @staticmethod
420     def ring_sub(a, b, m):
421         if isinstance(a, int):
422             a = BigInt(a)
423         if isinstance(b, int):
424             b = BigInt(b)
425         if isinstance(m, int):
```

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

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

```
508     def lsbgcd(a, b):
509         if isinstance(a, int):
510             a = BigInt(a)
511         if isinstance(b, int):
512             b = BigInt(b)
513         is_swap = False
514         if b > a:
515             a, b = b, a
516             is_swap = True
517         zero, one, two = BigInt(0), BigInt(1), BigInt(2)
518         x, y = a, b
519         A, B, C, D = one, zero, zero, one
520         log2_10 = BigInt(3)
521         while y:
522             n = log2_10 * BigInt(str(len(y.value)))
523             two_n = two.bipow(n)
524             left = two_n * y
525             right = two_n * two * y
526             while True:
527                 if left <= x < right:
528                     break
529                 if x < left:
530                     n = n - one
531                 if x >= right:
532                     n = n + one
533                 two_n = two.bipow(n)
534                 left = two_n * y
535                 right = two_n * two * y
536             s = x - left
537             p = right - x
538             if s <= p:
539                 t = s
540                 At = A - two_n * C
541                 Bt = B - two_n * D
542             else:
543                 t = p
544                 At = two_n * two * C - A
545                 Bt = two_n * two * D - B
546             if t <= y:
547                 x = y
548                 y = t
```

```
549                 A, B, C, D = C, D, At, Bt
550             else:
551                 x = t
552                 A = At
553                 B = Bt
554         if is_swap:
555             return x, B, A # d, v, u
556         return x, A, B # d, u, v
557 
558 
559 def GCD(a, b):
560     if a < 0:
561         a = -a
562     if b < 0:
563         b = -b
564     while b:
565         a, b = b, a % b
566     return a
567 
568 
569 def binary_GCD(num1, num2):
570     if num1 < 0:
571         num1 = -num1
572     if num2 < 0:
573         num2 = -num2
574     shift = 0
575     # Если одно из чисел равно нулю, делитель - другое число
576     if num1 == 0:
577         return num2
578     if num2 == 0:
579         return num1
580     # Если num1 = 1010, а num2 = 0100, то num1 | num2 = 1110
581     # 1110 & 0001 == 0, тогда происходит сдвиг, который фиксируется в shift
582     while (num1 | num2) \& 1 == 0:
583         shift += 1
584         num1 = num1 >> 1
585         num2 = num2 >> 1
586     # Если True, значит num1 - четное, иначе - нечетное
587     while num1 \& 1 == 0:
588         # если нечетное, сдвигаем на один бит
589         num1 = num1 >> 1
```

```
590     while num2 != 0:
591         # пока число нечётное, сдвигаем на один бит
592         while num2 \& 1 == 0:
593             num2 = num2 >> 1
594         # если первое число больше второго
595         if num1 > num2:
596             # меняем их местами
597             num1, num2 = num2, num1
598         # теперь первое число меньше второго, вычитаем
599         num2 = num2 - num1
600     # возвращаем число, перед этим сдвинув его биты на shift
601     return num1 << shift
602
603
604 if __name__ == '__main__':
605     while True:
606         menu_text = '\n'.join([
607             'Выберитедействие:',
608             '1) x + y',
609             '2) x - y',
610             '3) x * y',
611             '4) x / y',
612             '5) Возведениечислах встепеньу',
613             '6) Извлечениекорняизчислах степениу',
614             '7) НахождениеНОДх иу спомощьюалгоритмаЕвклида',
615             '8) НахождениеНОДх иу спомощьюбинарногоалгоритма',
616             '9) НахождениеНОДх иу, атакжеихкоэффициентыu, v спомощьюалгоритма
                LSBGCD',
617             'q) Выход'
618         ])
619         print(menu_text)
620         choice = input()
621         if choice == '1':
622             x = BigInt(input('Введите первоечисло(x): '))
623             y = BigInt(input('Введите второечисло(y): '))
624             t = time.time()
625             res = x + y
626             print(f'\nВремя расчета: {time.time() - t} сек.')
627             print('x + y =', res, '\n\n')
628         elif choice == '2':
629             x = BigInt(input('Введите первоечисло(x): '))
```

```
630             y = BigInt(input('Введите второечисло(y): '))
631             t = time.time()
632             res = x - y
633             print(f'\nВремя расчета: {time.time() - t} сек.')
634             print('x - y =', res, '\n\n')
635         elif choice == '3':
636             x = BigInt(input('Введите первоечисло(x): '))
637             y = BigInt(input('Введите второечисло(y): '))
638             t = time.time()
639             res = x * y
640             print(f'\nВремя расчета: {time.time() - t} сек.')
641             print('x * y =', res, '\n\n')
642         elif choice == '4':
643             x = BigInt(input('Введите первоечисло(x): '))
644             y = BigInt(input('Введите второечисло(y): '))
645             t = time.time()
646             res = x / y
647             print(f'\nВремя расчета: {time.time() - t} сек.')
648             print('x / y =', res, '\n\n')
649         elif choice == '5':
650             x = BigInt(input('Введите первоечисло(x): '))
651             y = int(input('Введите второечисло(y): '))
652             t = time.time()
653             res = x.bipow(y)
654             print(f'\nВремя расчета: {time.time() - t} сек.')
655             print('x**y =', res, '\n\n')
656         elif choice == '6':
657             x = BigInt(input('Введите первоечисло(x): '))
658             y = int(input('Введите второечисло(y): '))
659             t = time.time()
660             res = x.birt(y)
661             print(f'\nВремя расчета: {time.time() - t} сек.')
662             print('x**(1/y) =', res, '\n\n')
663         elif choice == '7':
664             x = BigInt(input('Введите первоечисло(x): '))
665             y = BigInt(input('Введите второечисло(y): '))
666             t = time.time()
667             res = GCD(x, y)
668             print(f'\nВремя расчета: {time.time() - t} сек.')
669             print('GCD(x, y) =', res, '\n\n')
670         elif choice == '8':
```

```
671             x = BigInt(input('Введите первоечисло(x): '))
672             y = BigInt(input('Введите второечисло(y): '))
673             t = time.time()
674             res = binary_GCD(x, y)
675             print(f'\nВремя расчета: {time.time() - t} сек.')
676             print('binary_GCD(x, y) =', res, '\n\n')
677         elif choice == '9':
678             x = BigInt(input('Введите первоечисло(x): '))
679             y = BigInt(input('Введите второечисло(y): '))
680             t = time.time()
681             d, u, v = BigInt.lsbgcd(x, y)
682             print(f'\nВремя расчета: {time.time() - t} сек.')
683             print(f'lsbgcd(x, y): d = {d}, u = {u}, v = {v}', '\n\n')
684         else:
685             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.