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 алгоритм и напишите соответствующую ему программу, позволяющую

- возводить целое число в квадрат;
- возводить натуральное число в натуральную степень;
- вычислять целую часть квадратного корня из натурального числа;
- вычислять целую часть кубического корня из натурального числа.

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», внутри которого находятся следующие методы:

- Выделение корня любой положительной целой степени из длинного целого числа.
  Программная реализация представляет подбор наиболее близкого числа, возведенного в данную из аргументов степень, при котором результат возведения в степень не будет превышать число, из которого выделяется корень. Выбор данного способа обусловлен простотой его реализации;

- Возведение в степень длинного целого числа.
  Программная реализация представляет умножение данного числа на самого себя, используя рекурсивные вызовы этой же функции. Размер этого повторного умножение равно числу, в степень которого необходимо возвести некоторое число. Выбор данного способа реализации обусловлено желанием опробовать рекурсию на практике.

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

```
1 a = BigInt('-12345678909876543 21')
2 b = BigInt('98765432101234567 89')
3 zero = BigInt()
```

- Выполняем возведение в степень:

```
1 print(a.bipow(20))
```

Вывод:
67654945781131788253399139476950939867213847384221510782372183
38736383554932818216005379411615896402318839463975841663187950
47266740645217094738013218419327830527872057771151857381511749
91352856101226216668855950857925749095871686783571452199421977
46524667992025003348862025953101533163689052346013027443912327
30289077240646312505876707771713512612446512064624 01

- Выполняем возведение в степень ноль:

```
1 print(b.bipow(0))
```

Вывод: 1

- Выполняем извлечение корня:

```
1 print(b.birt(9))
```

Вывод: 128

5 Вывод

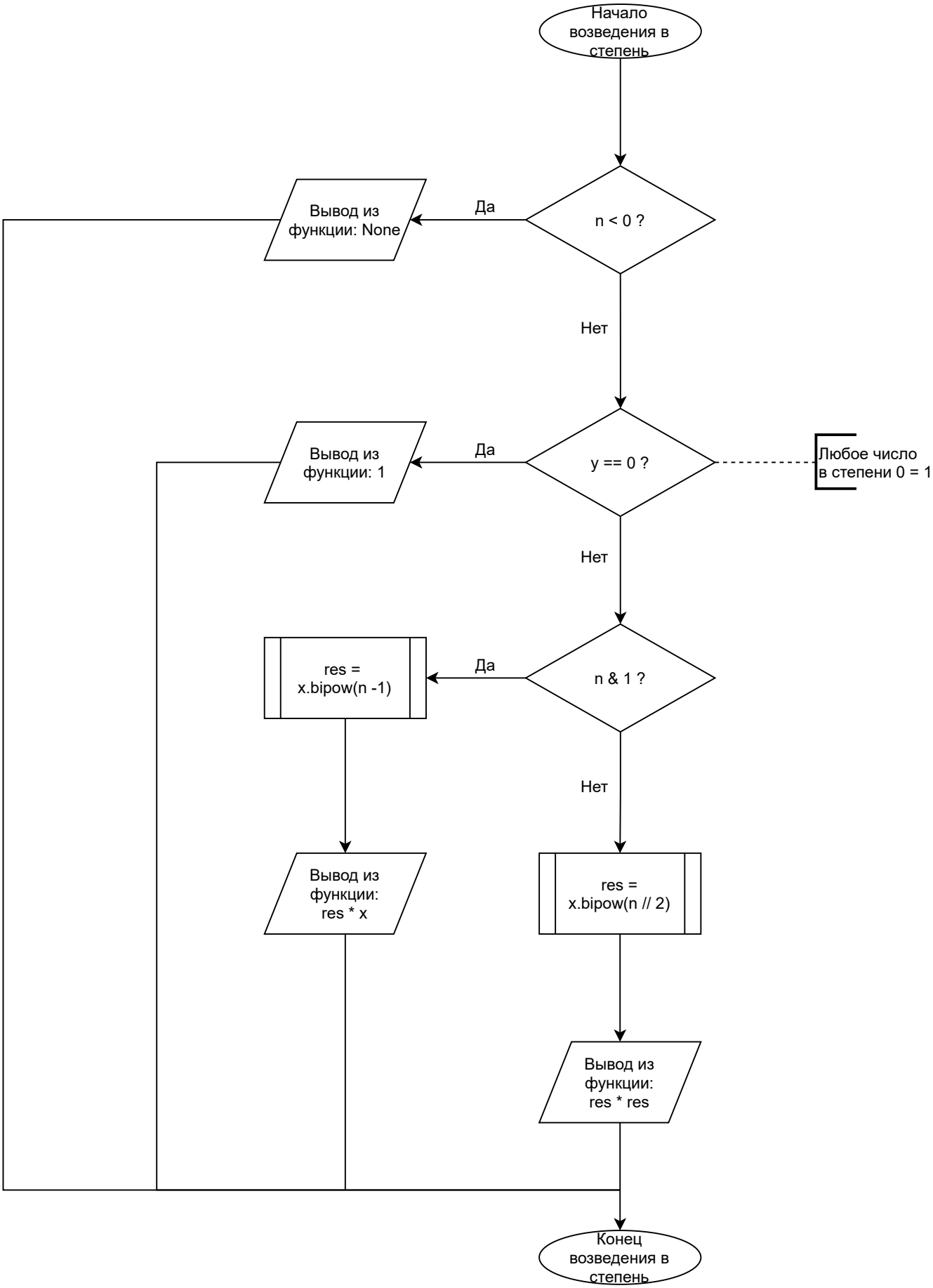
Мною был составлен алгоритм (в виде блок-схемы) и написана на языке Python программа, позволяющая:

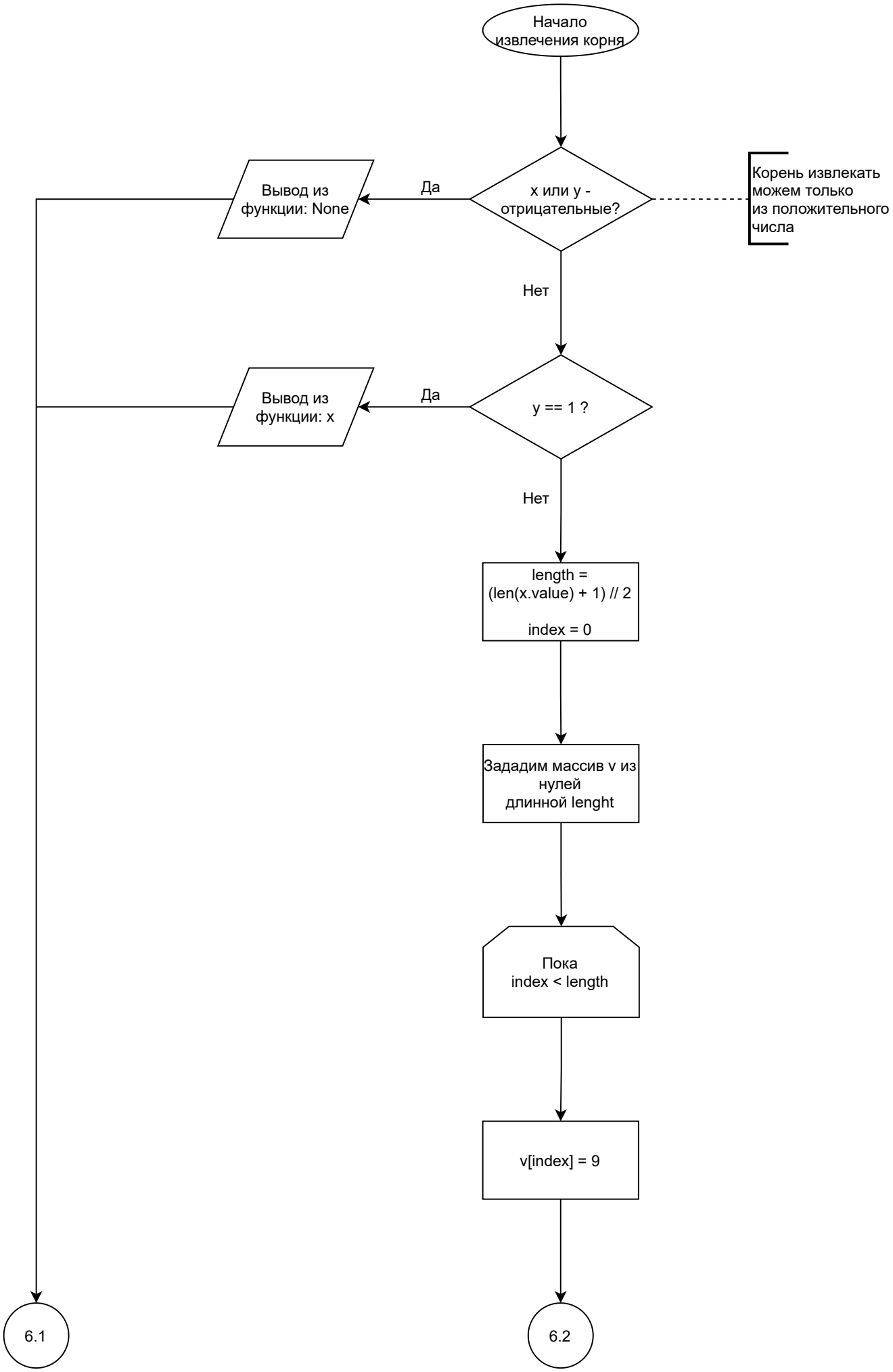
- возводить натуральное число в натуральную степень;
- вычислять целую часть n-го корня из натурального числа.

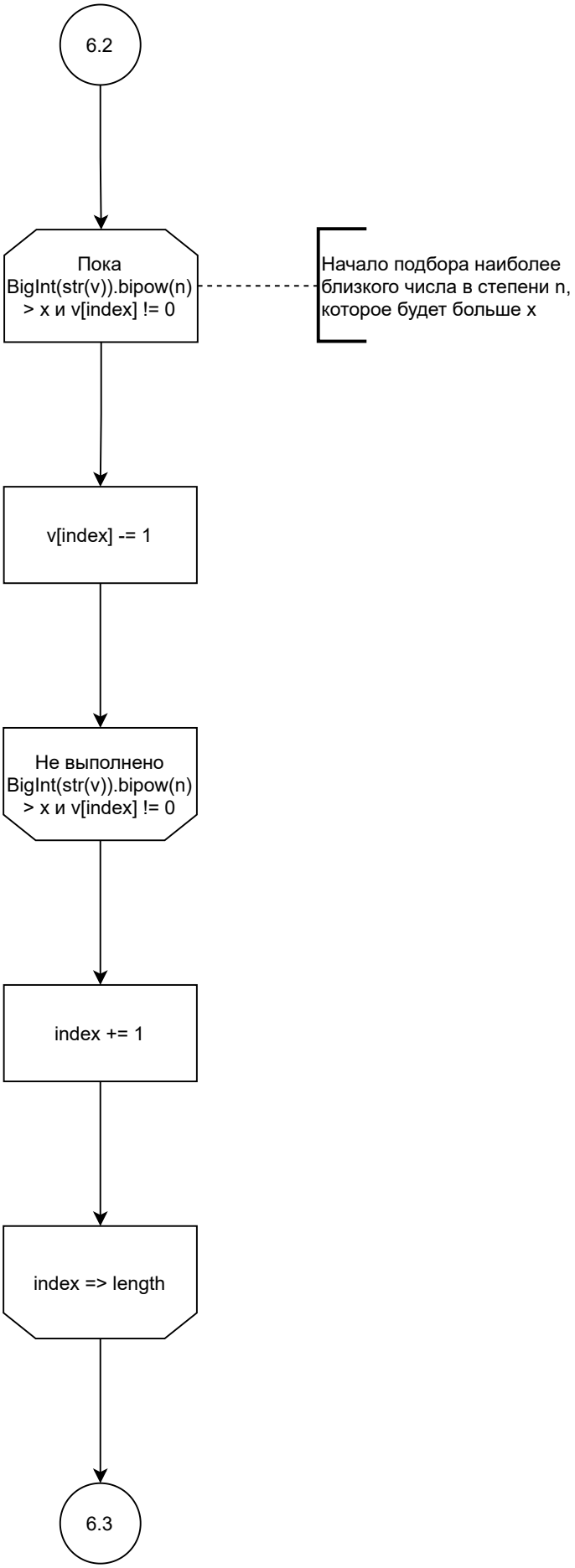
6 Блок-схема методов библиотеки

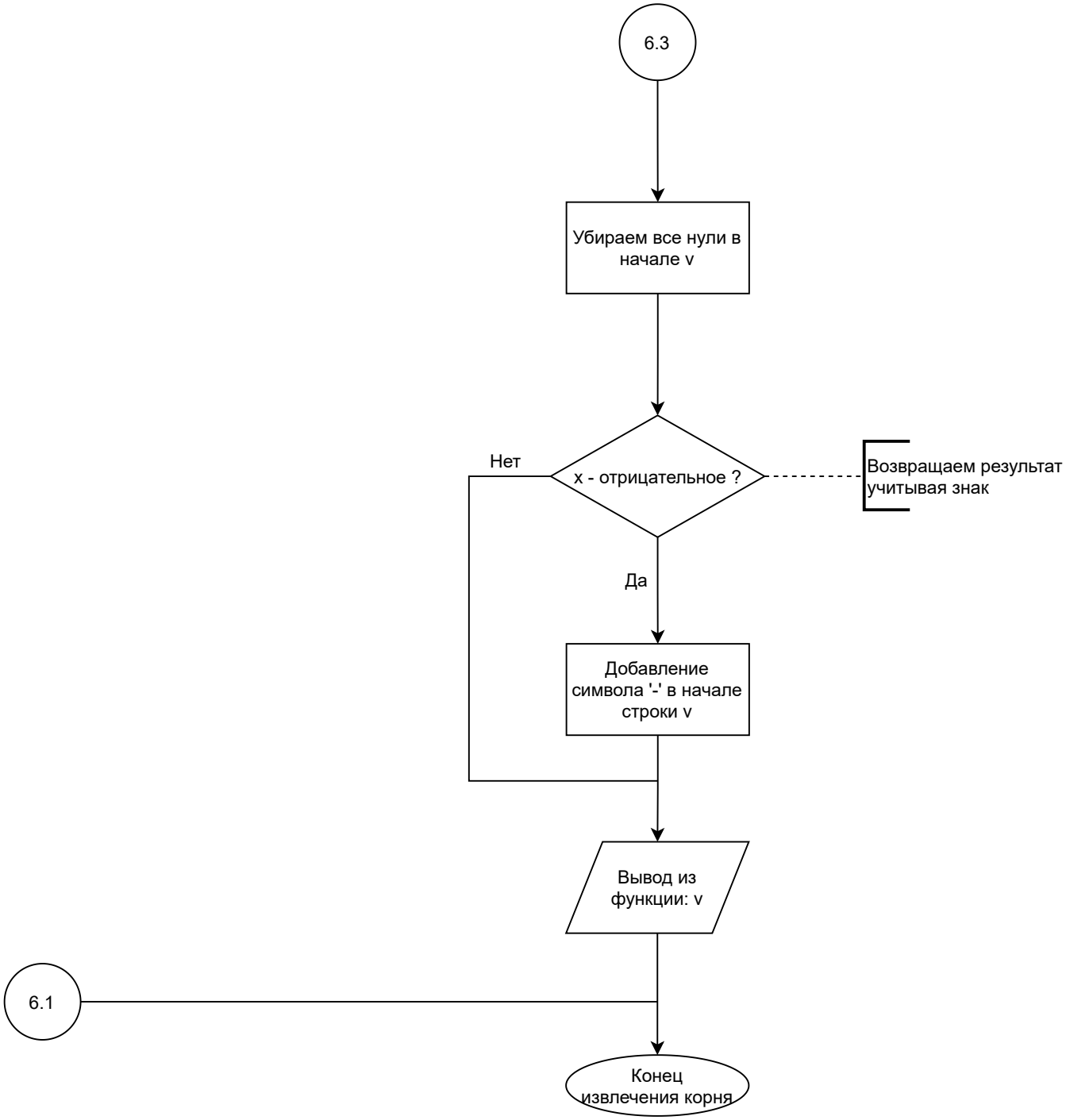
Ниже представлены блок-схемы методов в следующем порядке:

1. Метод возведения натурального числа в натуральную степень;
2. Метод вычисления целой части n-го корня из натурального числа.


Начало возведения в степень
n < 0 ?
Да
Вывод из функции: None
Нет
y == 0 ?
Да
Вывод из функции: 1
Любое число в степени 0 = 1
Нет
n & 1 ?
Да
res = x.bipow(n -1)
Нет
Вывод из функции: res * x
res = x.bipow(n // 2)
Вывод из функции: res * res
Конец возведения в степень


Начало
извлечения корня
x или y -
отрицательные?
Да
Вывод из
функции: None
Корень извлекать
можем только
из положительного
числа
Нет
y == 1 ?
Да
Вывод из
функции: x
Нет
length =
(len(x.value) + 1) // 2
index = 0
Зададим массив v из
нулей
длинной lenght
Пока
index < length
v[index] = 9
6.1
6.2


6.2
Пока BigInt(str(v)).bipow(n) > x и v[index] != 0
Начало подбора наиболее близкого числа в степени n, которое будет больше x
v[index] -= 1
Не выполнено BigInt(str(v)).bipow(n) > x и v[index] != 0
index += 1
index => length
6.3


6.3
Убираем все нули в начале v
x - отрицательное ?
Нет
Возвращаем результат учитывая знак
Да
Добавление символа '-' в начале строки v
Вывод из функции: v
6.1
Конец извлечения корня

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     """Классработысбольшимицелымичислами"""
11     is_neg = False # Флаг отрицательности числа
12     value = ''# Число в виде стоки
13
14     def __init__(self, x=0):
15         """Безаргументовконструкторзадаетзначениеравноенулю.
16         Если‘x‘ - число, конструкторзаполняетэкземплярэтимчислом.
17         Если‘x‘ - строка, котораямоглабыбытьчислом, конструкторзаполняетэкзе
               мплярэтимчислом.
18         Если‘x‘ - экземпляртогожекласса, конструкторкопируетегосодержимоевно
               выйэкземпляр."""
19         self.value = '0'
20         # Если в конструктор передано целое число
21         if isinstance(x, int):
22             self.is_neg = x < 0
23             self.value = str(x if x >= 0 else -x)
24         # Если в конструктор передана строка
25         elif isinstance(x, str):
26             # Если вдруг пришла пустая строка, пропускаем, оставляем value =
                   0
27             if len(x):
28                 self.is_neg = x[0] == '-'
29                 # Значением будет все, после минуса, если он был. И убираем
                       ведущие нули, если они были
30                 self.value = x[self.is_neg:].lstrip('0')
31                 # Проверяем, является ли строка числом, если нет, value = 0
32                 if not self.value.isdigit():
33                     self.value = '0'
34                 if self.value == '0':
35                     self.is_neg = False
36         # Если в конструктор передан экземпляр того же класса, копируем его
               содержимое
```

```
37          elif isinstance(x, BigInt):
38              self.value = x.value
39              self.is_neg = x.is_neg
40
41      # Является ли число четным
42      def is_even(self):
43          """Являетсяличислочетным"""
44          return not (int(self.value[-1]) \& 1)
45
46      # Перегрузка числа по модулю
47      def __abs__(self):
48          return BigInt(self.value)
49
50      def bipow(self, n):
51          """Возведениечиславстепень‘n‘"""
52          # Любое число в степени 0 = 1
53          if n < 0:
54              return None
55          if not n:
56              return BigInt(1)
57          # if n & 1:
58          # return BigInt(self.bipow(n - 1)) * self
59          # tmp = BigInt(self.bipow(n // 2))
60          # return BigInt(tmp * tmp)
61          res = self
62          for _ in range(n - 1):
63              res *= self
64          return res
65
66      def birt(self, n):
67          """Вычислениекорнястепени‘n‘ изчисла"""
68          # Корень извлекать можем только из положительного числа
69          if (n < 0) or self.is_neg:
70              return None
71          if n == 1:
72              return self
73          length = (len(self.value) + 1) // 2
74          index = 0
75          v = [0] * length
76          while index < length:
77              v[index] = 9
```

```
78              while BigInt(''.join(str(x) for x in v)).bipow(n) > self and v[
                    index]:
79                  v[index] -= 1
80              index += 1
81          v = ''.join(str(x) for x in v).lstrip('0')
82          return BigInt('-' + v) if self.is_neg else BigInt(v)
83
84      # Перегрузка перевода в bool
85      def __bool__(self):
86          return self.value != '0'
87
88      # Перегрузка x < y
89      def __lt__(self, other):
90          if isinstance(other, int):
91              other = BigInt(other)
92          self_len = len(self.value) # Запоминаем длину первого числа
93          self_other = len(other.value) # Запоминаем длину второго числа
94          # Если знаки одинаковые, то проверяем значения
95          if self.is_neg == other.is_neg:
96              # Если длины не равны
97              if self_len != self_other:
98                  # Меньше число с меньшей длинной для положительных и с
                        большей длиной для отрицательных
99                  return (self_len < self_other) ^ self.is_neg
100             i = 0
101             # Ищем разряд, в котором значения отличаются
102             while (i < self_len and self.value[i] == other.value[i]):
103                 i += 1
104             # Если разряд найден, то меньше число с меньшей цифрой для
                    положительных и с большей цифрой для отрицательных, иначе
                    числа равны
105             return (i < self_len) and ((self.value[i] < other.value[i]) ^
                    self.is_neg)
106         return self.is_neg # Знаки разные, если число отрицательное, то оно
                меньше, если положительное, то больше
107
108     # Перегрузка x <= y
109     def __le__(self, other):
110         return self < other or self == other
111
112     # Перегрузка x == y
113     def __eq__(self, other):
114         if isinstance(other, int):
```

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

```
156             return BigInt(0)
157         self_bin = self.to_bin()
158         return BigInt(int(self_bin + ('0' * n), 2))
159 
160     # Побитовое И (x & y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
161     def __and__(self, other):
162         if isinstance(other, int):
163             other = BigInt(other)
164         self_bin = self.to_bin()[2:]
165         other_bin = other.to_bin()[2:]
166         self_len = len(self_bin)
167         other_len = len(other_bin)
168         if self_len > other_len:
169             other_bin = other_bin.zfill(self_len)
170         elif self_len < other_len:
171             self_bin = self_bin.zfill(other_len)
172         res = int('0b' + ''.join(['1' if (x, y) == ('1', '1') else '0' for x,
                y in zip(self_bin, other_bin)]), 2)
173         return BigInt(res)
174 
175     # Побитовое ИЛИ (x | y) (ТОЛЬКО ПОЛОЖИТЕЛЬНЫЕ)
176     def __or__(self, other):
177         if isinstance(other, int):
178             other = BigInt(other)
179         self_bin = self.to_bin()[2:]
180         other_bin = other.to_bin()[2:]
181         self_len = len(self_bin)
182         other_len = len(other_bin)
183         if self_len > other_len:
184             other_bin = other_bin.zfill(self_len)
185         elif self_len < other_len:
186             self_bin = self_bin.zfill(other_len)
187         res = int('0b' + ''.join(['0' if (x, y) == ('0', '0') else '1' for x,
                y in zip(self_bin, other_bin)]), 2)
188         return BigInt(res)
189 
190     # Возврат копии
191     def copy(self):
192         """Возвраткопии"""
193         return BigInt(('-' if self.is_neg else '') + self.value)
194 
```

```
195     # Сложение двух чисел
196     def __add__(self, other):
197         if isinstance(other, int):
198             other = BigInt(other)
199         # Если знаки одинаковые, то выполняем сложение
200         if other.is_neg == self.is_neg:
201             num2 = other.value # Запоминаем значение второго операнда
202             self_len = len(self.value) # Длинна первого операнда
203             other_len = len(num2) # Длинна второго операнда
204             # Длина суммы равна максимуму из двух длин + 1 из-за возможного
                    переноса разряда
205             length = max(self_len, other_len)
206             res = [0] * (length + 1)
207             for i in range(length):
208                 j = length - i
209                 # Выполняем сложение разрядов
210                 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'))
211                 res[j - 1] = res[j] // 10 # Выполняем перенос в следующий
                        разряд, если он был
212                 res[j] = res[j] % 10 # Оставляем только единицы от возможного
                        переноса и превращаем символ в цифру
213                 # Возвращаем результат, учитывая его знак
214             return BigInt(('-' if self.is_neg else '') + ''.join(str(x) for x
                     in res))
215         # Если одно из чисел отрицательное, а другое положительное,
                отправляем на вычитание, меняя знак
216         return (self - (-BigInt(other))) if self.is_neg else (other - (-
                BigInt(self)))
217
218     # Вычитание одного числа из другого
219     def __sub__(self, other):
220         if isinstance(other, int):
221             other = BigInt(other)
222         # Если числа равны, считать не нужно
223         if self == other:
224             return BigInt(0)
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) %
                int(str('-' if other.is_neg else '') + other.value))
365     tmp = BigInt(other.value)
366     divider_length = len(other.value) # запоминаем длину делителя
367     # Если длина больше 8, то обнуляем long'овый делитель, иначе
          переводим строку в long
368     divider_v = 0 if divider_length > 8 else int(other.value)
369     length = len(self.value)
370     index = 0
371     mod2 = self.copy()
372     v = ''
373     mod = None
374     while BigInt(v) < tmp and index < length:
375         v = v + self.value[index]
376         index += 1
377     while True:
378         if BigInt(v) >= tmp:
379             if divider_v:
380                 v = str(int(v) % divider_v)
381             else:
382                 mod = BigInt(v)
```

```
383                     while mod >= tmp:
384                         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 
404 def GCD(a, b):
405     """Нахождениенаибольшегообщегоделителяучисел‘a‘ и‘b‘"""
406     if a < 0:
407         a = -a
408     if b < 0:
409         b = -b
410     while b:
411         a, b = b, a % b
412     return a
413 
414 
415 def binary_GCD(num1, num2):
416     """Нахождениенаибольшегообщегоделителяучисел‘a‘ и‘b‘ (Бинарный алгоритм)
        """
417     if num1 < 0:
418         num1 = -num1
419     if num2 < 0:
420         num2 = -num2
421     shift = 0
422     # Если одно из чисел равно нулю, делитель - другое число
```

```
423     if num1 == 0:
424         return num2
425     if num2 == 0:
426         return num1
427     # Если num1 = 1010, а num2 = 0100, то num1 | num2 = 1110
428     # 1110 & 0001 == 0, тогда происходит сдвиг, который фиксируется в shift
429     while (num1 | num2) \& 1 == 0:
430         shift += 1
431         num1 = num1 >> 1
432         num2 = num2 >> 1
433     # Если True, значит num1 - четное, иначе - нечетное
434     while num1 \& 1 == 0:
435         # если нечетное, сдвигаем на один бит
436         num1 = num1 >> 1
437     while num2 != 0:
438         # пока число нечётное, сдвигаем на один бит
439         while num2 \& 1 == 0:
440             num2 = num2 >> 1
441         # если первое число больше второго
442         if num1 > num2:
443             # меняем их местами
444             num1, num2 = num2, num1
445         # теперь первое число меньше второго, вычитаем
446         num2 = num2 - num1
447     # возвращаем число, перед этим сдвинув его биты на shift
448     return num1 << shift
449
450
451 if __name__ == '__main__':
452     while True:
453         menu_text = '\n'.join([
454             'Выберитедействие:',
455             '1) x + y',
456             '2) x - y',
457             '3) x * y',
458             '4) x / y',
459             '5) Возведениечислах встепеньу',
460             '6) Извлечениекорняизчислах степениу',
461             'q) Выход'
462         ])
463         print(menu_text)
```

```
464         choice = input()
465         if choice == '1':
466             x = BigInt(input('Введите первоечисло(x): '))
467             y = BigInt(input('Введите второечисло(y): '))
468             t = time.time()
469             res = x + y
470             print(f'\nВремя расчета: {time.time() - t} сек.')
471             print('x + y =', res, '\n\n')
472         elif choice == '2':
473             x = BigInt(input('Введите первоечисло(x): '))
474             y = BigInt(input('Введите второечисло(y): '))
475             t = time.time()
476             res = x - y
477             print(f'\nВремя расчета: {time.time() - t} сек.')
478             print('x - y =', res, '\n\n')
479         elif choice == '3':
480             x = BigInt(input('Введите первоечисло(x): '))
481             y = BigInt(input('Введите второечисло(y): '))
482             t = time.time()
483             res = x * y
484             print(f'\nВремя расчета: {time.time() - t} сек.')
485             print('x * y =', res, '\n\n')
486         elif choice == '4':
487             x = BigInt(input('Введите первоечисло(x): '))
488             y = BigInt(input('Введите второечисло(y): '))
489             t = time.time()
490             res = x / y
491             print(f'\nВремя расчета: {time.time() - t} сек.')
492             print('x / y =', res, '\n\n')
493         elif choice == '5':
494             x = BigInt(input('Введите первоечисло(x): '))
495             y = int(input('Введите второечисло(y): '))
496             t = time.time()
497             res = x.bipow(y)
498             print(f'\nВремя расчета: {time.time() - t} сек.')
499             print('x**y =', res, '\n\n')
500         elif choice == '6':
501             x = BigInt(input('Введите первоечисло(x): '))
502             y = int(input('Введите второечисло(y): '))
503             t = time.time()
504             res = x.birt(y)
```

```
505             print(f'\nВремя расчета: {time.time() - t} сек.')
506             print('x**(1/y) =', res, '\n\n')
507         else:
508             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.