МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Компьютерная практика

Выполнил: ст. группы ПВ-232
Чернобровнеко А.Е.
Проверил: Солонченко Р.Е.

Белгород, 2024г.

Оглавление

1 Задания к работе **3**

2 Основная часть **5**

2.1 Тема 1. Линейные алгоритмы. Задание 23. 5

2.2 Тема 2. Разветвляющиеся алгоритмы. Задание 23. 7

2.3 Тема 3. Циклические и итерационные алгоритмы. Задание 23. 10

2.4 Тема 4. Простейшие операции над массивами. Задание 23. 12

2.5 Тема 5. Векторы и матрицы. Задание 23. 14

2.6 Тема 6. Линейный поиск. Задание 23. 17

2.7 Тема 7. Разветвляющиеся алгоритмы. Задание 23. 22

2.8 Тема 8. Геометрия и теория множеств. Задание 23. 27

2.9 Тема 9. Линейная алгебра и сжатие информации. Задание 23. 32

1 Задания к работе

Вариант 23

1. Текущее время (часы, минуты, секунды) задано тремя переменными: *h, m, s*. Округлить его до целых значений минут и часов. Например, 14 ч 21 мин 45 с преобразуется в 14 ч 22 мин или 14 ч, а 9 ч 59 мин 23 с — соответственно в 9 ч 59 мин или 10 ч.

2. Для заданного $0 < n \leq 200$, рассматриваемого как возраст человека, вывести фразу вида: «Мне 21 год», «Мне 32 года», «Мне 12 лет».

3. Леспромхоз ведёт заготовку деловой древесины. Первоначальный объём её на территории леспромхоза составлял p кубометров. Ежегодный прирост составляет $k\%$. Годовой план заготовки — t кубометров. Через сколько лет в бывшем лесу будут расти одни опята?

4. Каждый из элементов x_i массива *X(n)* заменить средним значением первых i элементов этого массива.

5. Многочлены $P_n(x)$ и $Q_m(x)$ заданы своими коэффициентами. Определить коэффициенты их композиции — многочлена $P_n(Q_m(x))$.

6. В массиве *P(n)* найти самую длинную последовательность, которая является арифметической или геометрической прогрессией.

7. Найти все натуральные числа, не превосходящие заданного n, десятичная запись которых есть строго возрастающая или строго убывающая последовательность цифр.

8. Медианой множества точек на плоскости назовём прямую, которая делит множество на два подмножества одинаковой мощности. Найти горизонтальную и вертикальную медианы заданного множества, у которого никакие две точки не лежат на одной горизонтальной или вертикальной прямой.

9. Заданный неупакованный двоичный массив сжать, используя полубайтовое представление длин цепочек.

10. По правилам пунктуации пробел может стоять после, а не перед каждым из следующих знаков: . , ; : ! ?)] } . . . ; перед, а не после знаков: ([{. Заданный текст проверить на соблюдение этих правил и при необходимости исправить. Вместо пробела может быть перевод строки или знак табуляции.

11. Построить прямую $3x + 2y - 4 = 0$ в диапазоне $x \in [-1; 3]$ с шагом $\Delta = 0,25$

12. Построить верхнюю часть эллипса $0,1 \leq x \leq 5,1$ с шагом $\Delta = 0,25$, заданного уравнением $\frac{x^2}{4} + y^2 = 1$

13. Найдите точку равновесия в заданном диапазоне с заданным шагом.

$$\begin{cases} y = \frac{2}{x} & \text{в диапазоне } 0.1 \leq x \leq 4, \text{ с шагом } \Delta = 0.1 \\ y^2 = 2x \end{cases}$$

14. Построить плоскость, проходящую через точки $M_1(3,3,1)$, $M_2(2,3,2)$, $M_3(1,1,3)$, при $-1 \leq x \leq 4$ с шагом $\Delta = 0,5$ и $-1 \leq y \leq 3$ с шагом $\Delta = 1$.

15. Построить часть параболоида, заданного уравнением $\frac{x^2}{9} + \frac{y^2}{4} = 2z$, лежащую в диапазоне $-3 \leq x \leq 3$, $-2 \leq y \leq 2$ с шагом $\Delta = 0,5$ для обеих переменных.

2 Основная часть

2.1 Тема 1. Линейные алгоритмы. Задание 23.

1. Текущее время (часы, минуты, секунды) задано тремя переменными: *h, m, s*. Округлить его до целых значений минут и часов. Например, 14 ч 21 мин 45 с преобразуется в 14 ч 22 мин или 14 ч, а 9 ч 59 мин 23 с — соответственно в 9 ч 59 мин или 10 ч.

2. Словесное описание *алгоритма*:

   (a) Знаем, что в одном часе 60 минут, а в одной минуте 60 секунд. Будем придерживаться принципа округления - если количество секунд $< 50\%$, тогда округляем количество минут в меньшую сторону, иначе - в большую. С минутами в часах будем действовать по тому же принципу.

   (b) Исходя из вышесказанного, если количество секунд < 30, тогда округляем минуты в меньшую сторону, иначе - в большую. С минутами в часах действуем по тому же принципу.

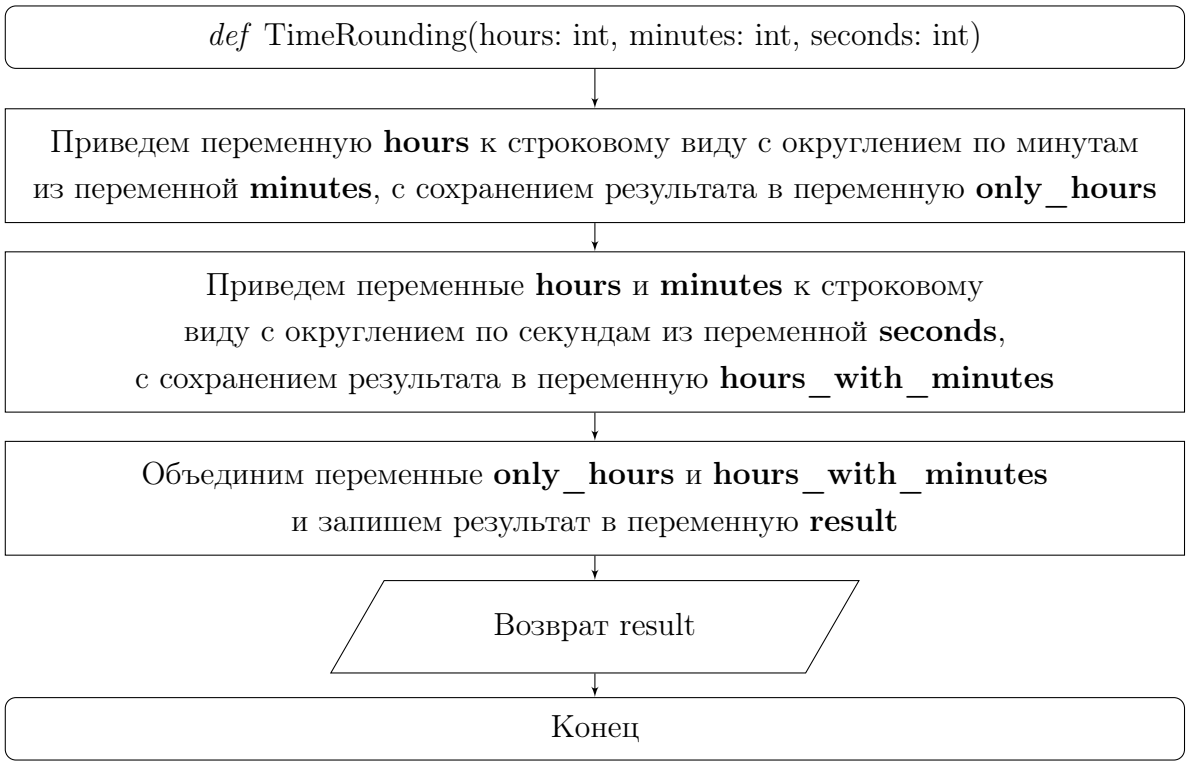
   (c) Для решения сначала запишем результаты округлений в соответствующие переменные, а потом объединим их в общую.

3. Спецификация функции *TimeRounding*:

   (a) Заголовок: `def TimeRounding(hours: int, minutes: int, seconds: int) -> str:`

   (b) Назначение: используется для нахождения целого числа минут и часов или только часов по введенному времени.

Блок-схема:



4. Код *алгоритма* на языке *Python*:

```python
def TimeRounding(hours: int, minutes: int, seconds: int) -> str:
    only_hours: str = f'{hours + 1 if minutes >= 30 else hours} ч'
    hours_with_minutes: str = \
        f'{hours} ч {minutes + 1 if seconds >= 30 else minutes} м'
    result: str = f'{hours_with_minutes} или {only_hours}'

    return result
```

5. *Таблица 1*

Тестовые данные

Входные данные	Выходные данные
14 21 45	14 ч 22 м или 14 ч
9 59 23	9 ч 59 м или 10 ч

2.2 Тема 2. Разветвляющиеся алгоритмы. Задание 23.

1. Для заданного $0 < n \leq 200$, рассматриваемого как возраст человека, вывести фразу вида: «Мне 21 год», «Мне 32 года», «Мне 12 лет».

2. Словесное описание *алгоритма*:

   (a) Если $n\%10 = 0$, $n\%10 \geq 5$ или $10 \leq n \leq 20$ - мы пишем «лет». Если n оканчивается на 1, при этом $n \neq 11$, тогда мы пишем «год». В остальных случаях мы пишем «года».

   (b) Исходя из вышесказанного, запишем условия для получения правильной формы слова.

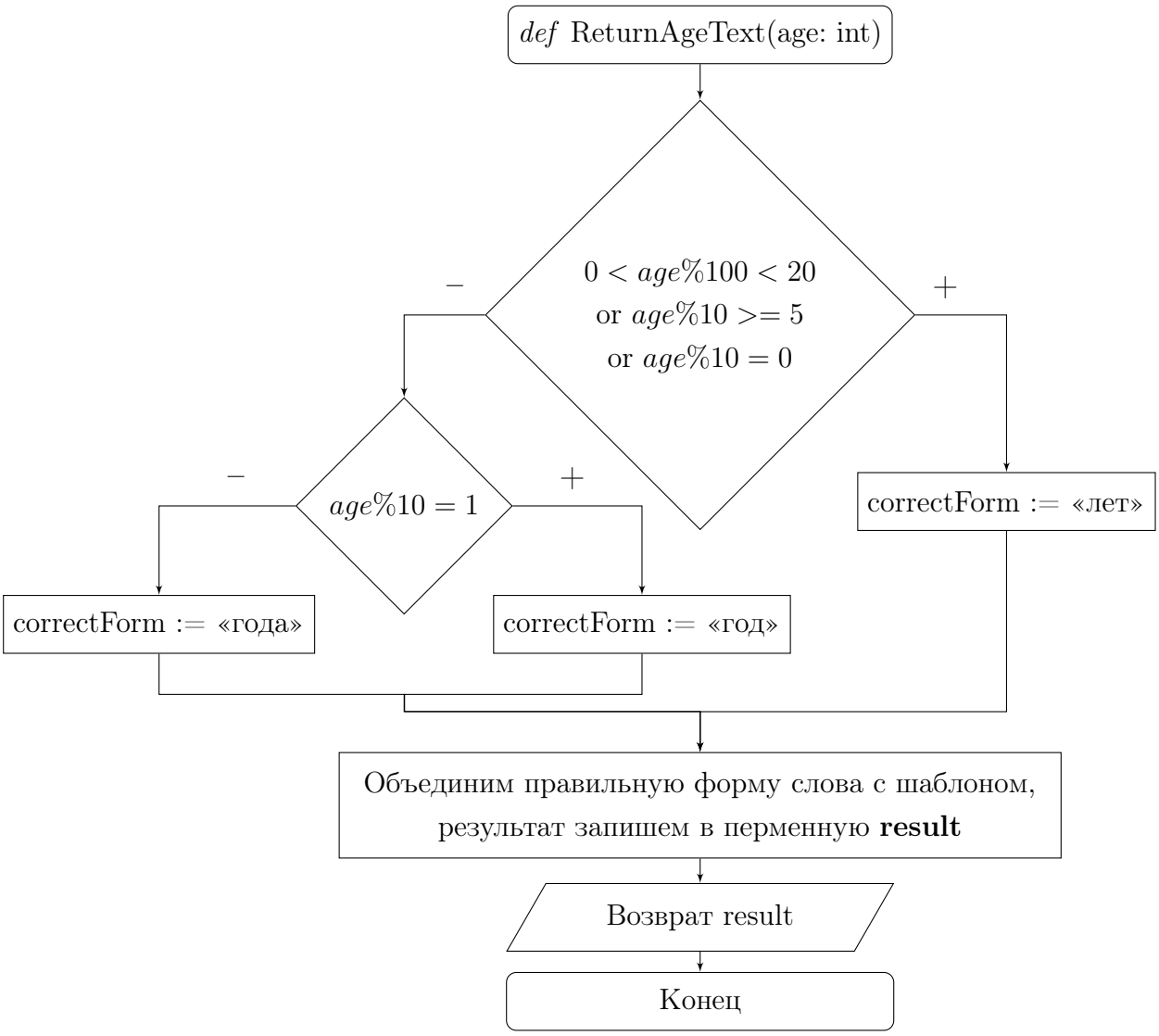
   (c) Получив правильную форму слова, объединим ее с шаблоном предложения.

3. Спецификация функции *ReturnAgeText*:

   (a) Заголовок: `def ReturnAgeText(age: int) -> str`

   (b) Назначение: используется для нахождения правильной формы слова «лет», обозначающего возраст, в шаблон «Мне n лет».

Блок-схема:



4. Код *алгоритма* на языке *Python*:

```python
def ReturnAgeText(age: int) -> str:
    if (10 < age % 100 < 20) or (age % 10 >= 5) or (age % 10 == 0):
        correctForm: str = 'лет'
    elif age % 10 == 1:
        correctForm: str = 'год'
    else:
        correctForm: str = 'года'
    answer: str = f'Мне {age} {correctForm}'

    return answer
```

5. *Таблица 2*

Тестовые данные

Входные данные	Выходные данные
11	Мне 11 лет
1	Мне 1 год
21	Мне 21 год
30	Мне 30 лет
42	Мне 42 года

2.3 Тема 3. Циклические и итерационные алгоритмы. Задание 23.

1. Леспромхоз ведёт заготовку деловой древесины. Первоначальный объём её на территории леспромхоза составлял p кубометров. Ежегодный прирост составляет $k\%$. Годовой план заготовки — t кубометров. Через сколько лет в бывшем лесу будут расти одни опята?

2. Словесное описание *алгоритма*:

   (a) Известно, что ежемесячно объем древесины p кубометров увеличивается на $k\%$, при этом уменьшался на t кубометров ежегодно.

   (b) Исходя из условия, самый простой способ узнать срок жизни леса - простой перебор, который будет считать, сколько древесины остается спустя каждый год.

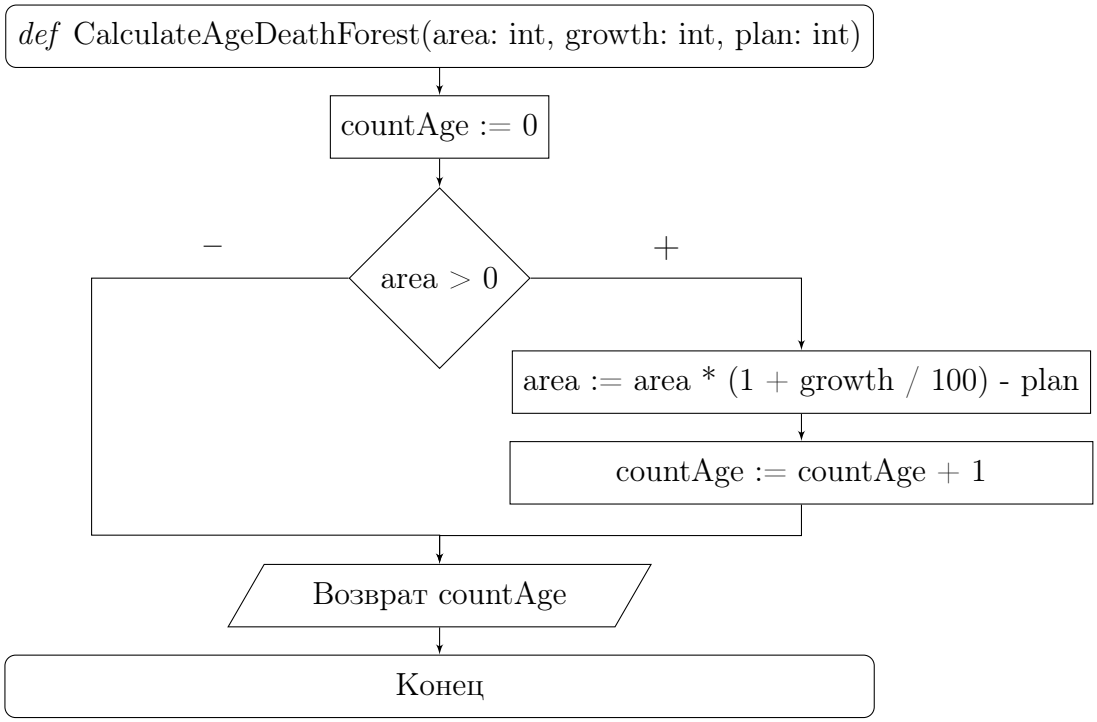
   (c) Организуем цикл, который будет отслеживать количество древесины спустя каждый год. Если количество ее кубометров будет = 0, значит срок жизни леса подошел к концу.

3. Спецификация функции *CalculateAgeDeathForest*:

   (a) Заголовок: `def CalculateAgeDeathForest(area: int, growth: int, plan: int) -> int`

   (b) Назначение: используется для нахождения срока жизни леса в годах.

   Блок-схема:



4. Код *алгоритма* на языке *Python*:

```python
def CalculateAgeDeathForest(area: int, growth: int, plan: int) -> int:
    countAge: int = 0
    while area > 0:
        area = area * (1 + growth / 100) - plan
        countAge += 1

    return countAge
```

5. *Таблица 3*

Тестовые данные

Входные данные	Выходные данные
10 9 4	3
10 10 4	4
1 100 2	1
0 1000 1	0

2.4 Тема 4. Простейшие операции над массивами. Задание 23.

1. Каждый из элементов x_i массива $X(n)$ заменить средним значением первых i элементов этого массива.

2. Словесное описание *алгоритма*:

   (a) Для того, чтобы заменить каждый элемент массива средним значением первых i элементов массива, необходимо найти это среднее значение.

   (b) Просуммируем первые i элементов массива и разделим сумму на i, чтобы получить их среднее значение.

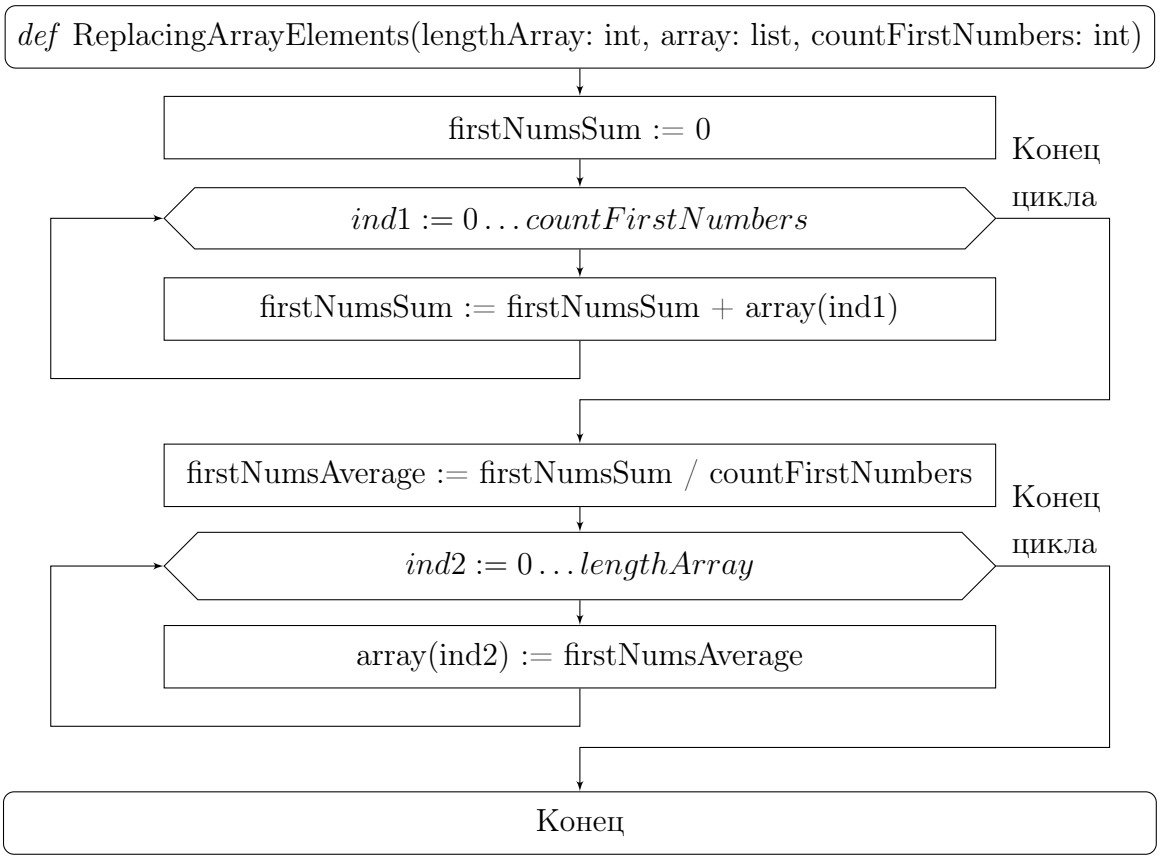
   (c) Имея среднее значение первых i элементов массива, заменим каждый его член на это значение.

3. Спецификация функции *ReplacingArrayElements*:

   (a) Заголовок:
   ```
   def ReplacingArrayElements(lengthArray: int, array: list,
   countFirstNumbers: int) -> None:
   ```

   (b) Назначение: используется для замены элементов массива на среднее значение его первых i элементов.

   Блок-схема:



4. Код *алгоритма* на языке *Python*:

```python
def ReplacingArrayElements(lengthArray: int, array: list,
                           countFirstNumbers: int) -> None:
    if (countFirstNumbers > lengthArray) | (countFirstNumbers == 0):
        return

    firstNumsSum: int = 0
    for ind in range(0, countFirstNumbers):
        firstNumsSum += array[ind]

    firstNumsAverage: float = firstNumsSum / countFirstNumbers

    for ind in range(0, lengthArray):
        array[ind] = firstNumsAverage
```

5. *Таблица 4*

Тестовые данные

Входные данные	Выходные данные
5 1 2 3 4 5 6	1 2 3 4 5
5 1 2 3 4 5 3	2 2 2 2 2
5 1 2 3 4 5 0	1 2 3 4 5

2.5 Тема 5. Векторы и матрицы. Задание 23.

1. Многочлены $P_n(x)$ и $Q_m(x)$ заданы своими коэффициентами. Определить коэффициенты их композиции — многочлена $P_n(Q_m(x))$.

2. Словесное описание *алгоритма*:

   (a) Для начала, чтобы получить коэффициенты композиции $P_n(Q_m(x))$, нам необходимо подставить на место x в многочлен $P_n(x)$ многочлен $Q_m(x)$.

   (b) После замены переменной x нам необходимо лишь выполнить вычисления для упрощения многочлена.

   (c) Вышесказанного достаточно, чтобы получить композицию многочленов и выразить ее коэффициенты. Существуют два частных случая:

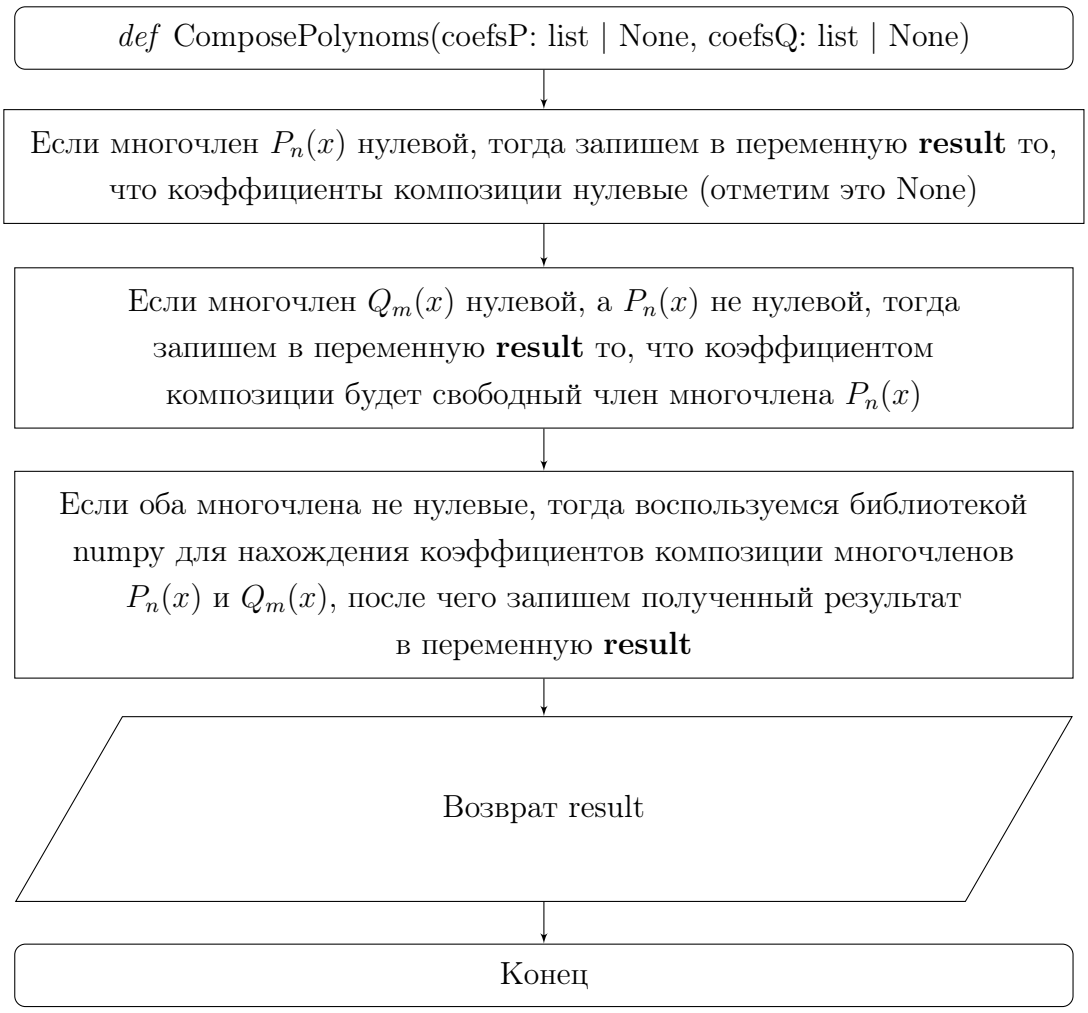
      1. Если многочлен $P_n(x)$ нулевой - тогда и композиция будет нулевой.
      2. Если многочлен $Q_m(x)$ нулевой - тогда у композиции будет один коэффициент, равный свободному члену многочлена $P_n(x)$.

3. Спецификация функции *ComposePolynomss*:

   (a) Заголовок:
   ```
   def ComposePolynoms(coefsP: list | None,
   coefsQ: list | None) -> list | None
   ```

   (b) Назначение: используется для нахождения коэффициентов композиции двух многочленов.

Блок-схема:



4. Код *алгоритма* на языке *Python*:

```python
def ComposePolynoms(coefsP: list | None, coefsQ: list | None) -> list | None:
    if coefsP is None:
        result: None = None
    elif coefsQ is None:
        result: list = [coefsP[-1]]
    else:
        PolynomP = np.poly1d(coefsP)
        PolynomQ = np.poly1d(coefsQ)
        composedPolynom = PolynomP(PolynomQ)
        result: list = composedPolynom.coefficients.tolist()

    return result
```

5. *Таблица 5*

Тестовые данные

Входные данные	Выходные данные
2 0 -5 1 -1 2	2 -4 10 -8 3
1 2 2 -5	2 -3
1 2 0 11	1
None 2 0 11	None
2 0 11 None	11

2.6 Тема 6. Линейный поиск. Задание 23.

1. В массиве *P(n)* найти самую длинную последовательность, которая является арифметической или геометрической прогрессией.

2. Словесное описание *алгоритма*:

   (a) Так как любые два числа образуют прогрессию, получаем коэффициент для поиска прогрессии от них.

   (b) Идем по элементам до тех пор, пока соблюдаются условия прогрессии.

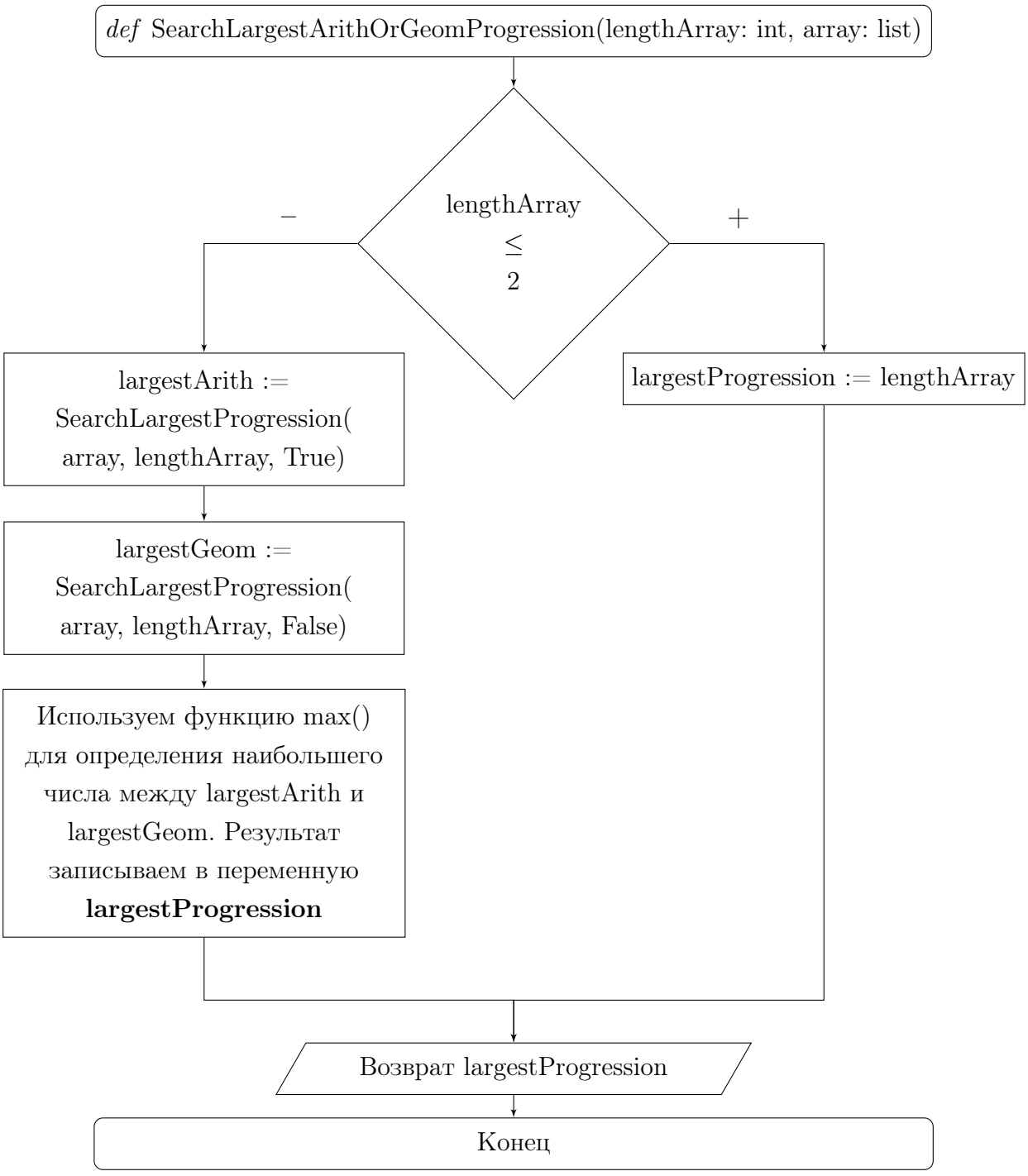
   (c) Когда прогрессия прерывается, берем ее последние два числа, получаем новый коэффициент и ищем новую прогрессию.

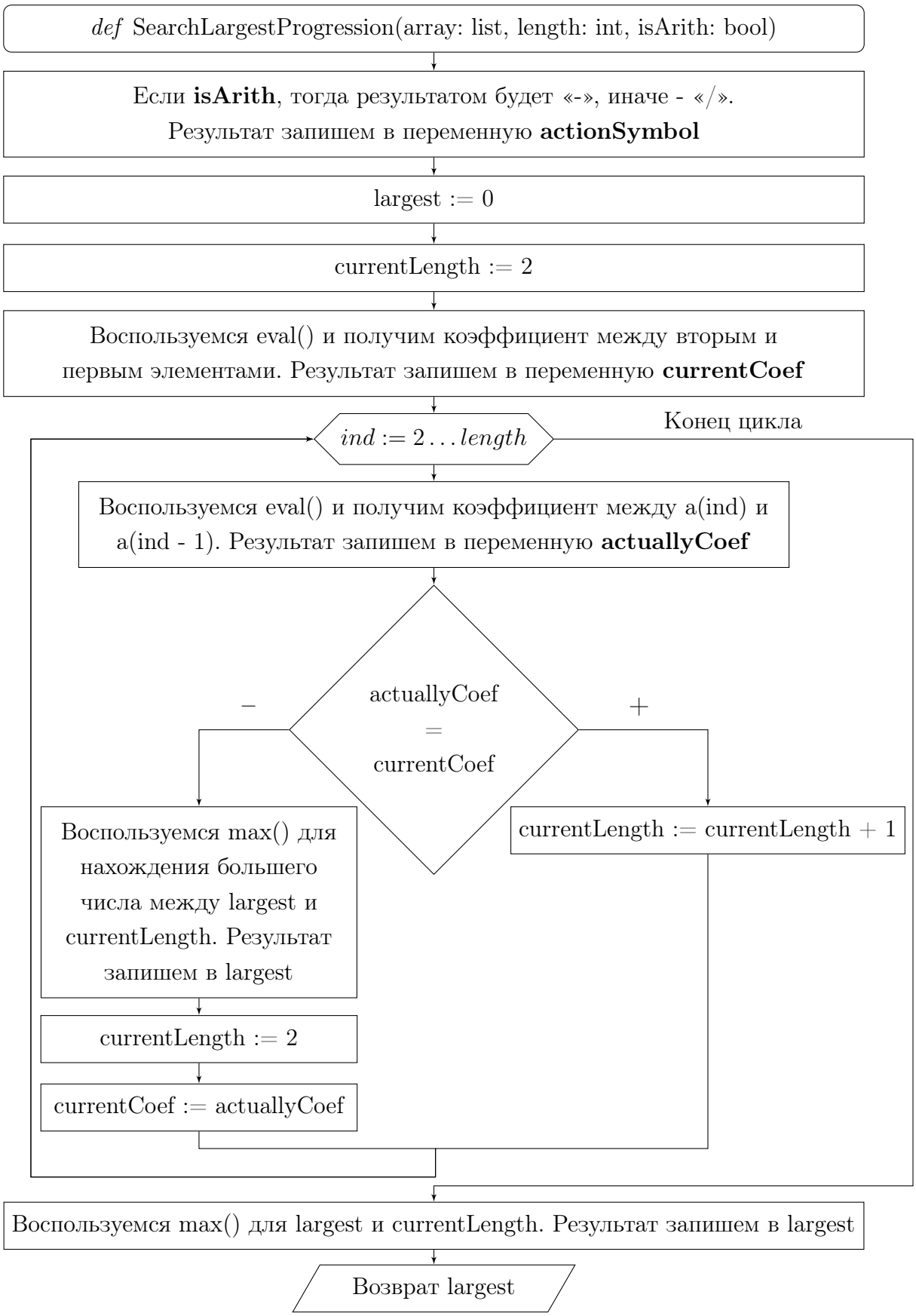
3. Спецификация функции *SearchLargestArithOrGeomProgression*:

   (a) Заголовок: *def* SearchLargestArithOrGeomProgression(lengthArray: int, array: list) -> int

   (b) Назначение: используется для нахождения наибольшей длины алгоритмической или геометрической прогрессии в массиве *P(n)*.

Блок-схемы:


def SearchLargestArithOrGeomProgression(lengthArray: int, array: list)
lengthArray ≤ 2
−
+
largestArith := SearchLargestProgression(array, lengthArray, True)
largestGeom := SearchLargestProgression(array, lengthArray, False)
Используем функцию max() для определения наибольшего числа между largestArith и largestGeom. Результат записываем в переменную largestProgression
largestProgression := lengthArray
Возврат largestProgression
Конец


def SearchLargestProgression(array: list, length: int, isArith: bool)
Если isArith, тогда результатом будет «-», иначе - «/».
Результат запишем в переменную actionSymbol
largest := 0
currentLength := 2
Воспользуемся eval() и получим коэффициент между вторым и первым элементами. Результат запишем в переменную currentCoef
ind := 2 . . . length
Конец цикла
Воспользуемся eval() и получим коэффициент между a(ind) и a(ind - 1). Результат запишем в переменную actuallyCoef
actuallyCoef = currentCoef
−
+
Воспользуемся max() для нахождения большего числа между largest и currentLength. Результат запишем в largest
currentLength := 2
currentCoef := actuallyCoef
currentLength := currentLength + 1
Воспользуемся max() для largest и currentLength. Результат запишем в largest
Возврат largest

4. Код *алгоритма* на языке *Python*:

```python
def SearchLargestArithOrGeomProgression(lengthArray: int, array: list) -> int:
    if lengthArray <= 2:
        largestProgression: int = lengthArray
    else:
        largestArith: int = SearchLargestProgression(
            array, lengthArray, True)
        largestGeom: int = SearchLargestProgression(
            array, lengthArray, False)
        largestProgression: int = max(largestArith, largestGeom)

    return largestProgression


def SearchLargestProgression(array: list, length: int, isArith: bool) -> int:
    actionSymbol: str = '-' if isArith else '/'
    largest: int = 0
    currentLength: int = 2
    currentCoef: float = eval(f'{array[1]} {actionSymbol} {array[0]}')
    for ind in range(2, length):
        actuallyCoef = eval(f'{array[ind]} {actionSymbol} {array[ind - 1]}')

        if actuallyCoef == currentCoef:
            currentLength += 1
        else:
            largest = max(largest, currentLength)
            currentLength = 2
            currentCoef = actuallyCoef

    largest = max(largest, currentLength)

    return largest
```

5. *Таблица 6*

Тестовые данные

Входные данные	Выходные данные
5 1 3 5 7 9	5
5 2 1 3 5 9	3
7 2 4 1 3 9 27 7	4
4 3 4 11 2	2
5 1 2 4 8 16	5
2 1 12	2
6 8 9 10 12 14 16	4

2.7 Тема 7. Разветвляющиеся алгоритмы. Задание 23.

1. Найти все натуральные числа, не превосходящие заданного n, десятичная запись которых есть строго возрастающая или строго убывающая последовательность цифр.

2. Словесное описание *алгоритма*:

   (a) Идея решения заключается в том, чтобы рекурсивно составлять числа, добавляя к существующему цифры, и проверять их на соответствие условиям задачи.

   (b) Для реализации озвученной выше идеи создадим цикл для старта составления чисел с определенной цифры.

   (c) Определенную цифру подаем в рекурсивную функцию, которая проверяет, не превышает ли оно заданного и является ли его десятичная запись строго возрастающей или строго убывающей последовательностью цифр.

   (d) Соответственно, если число будет подходить - тогда добавляем его к общей сумме таких чисел и продолжаем его составление дальше, в противном случае - заканчиваем эту рекурсивную ветвь.

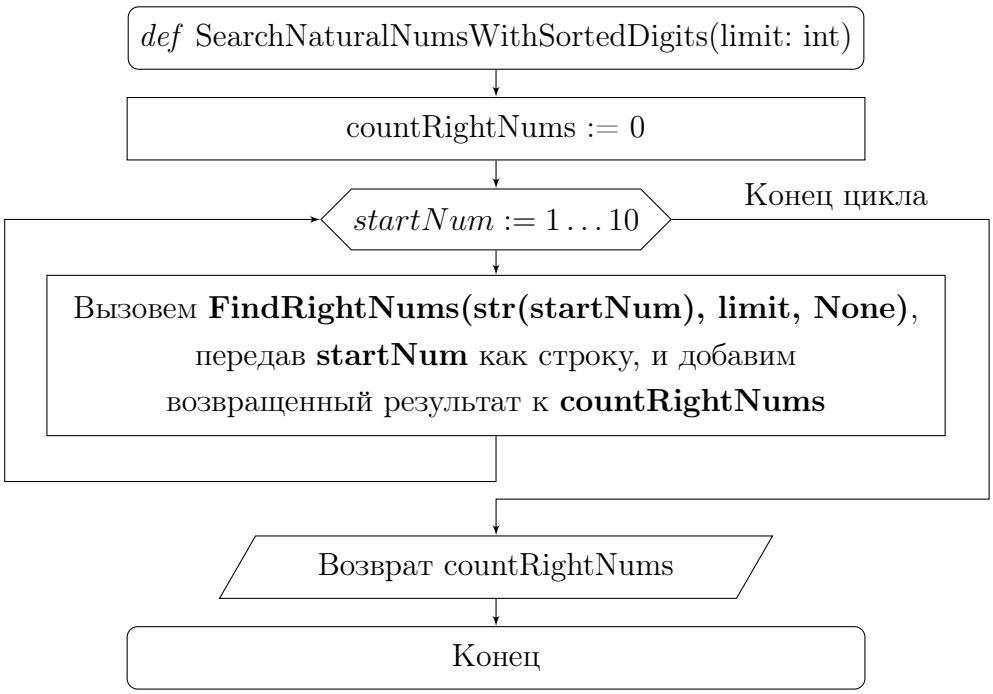
   (e) В конце нужно будет лишь просуммировать количество подходящих чисел от каждой начальной цифры и вернуть результат пользователю.

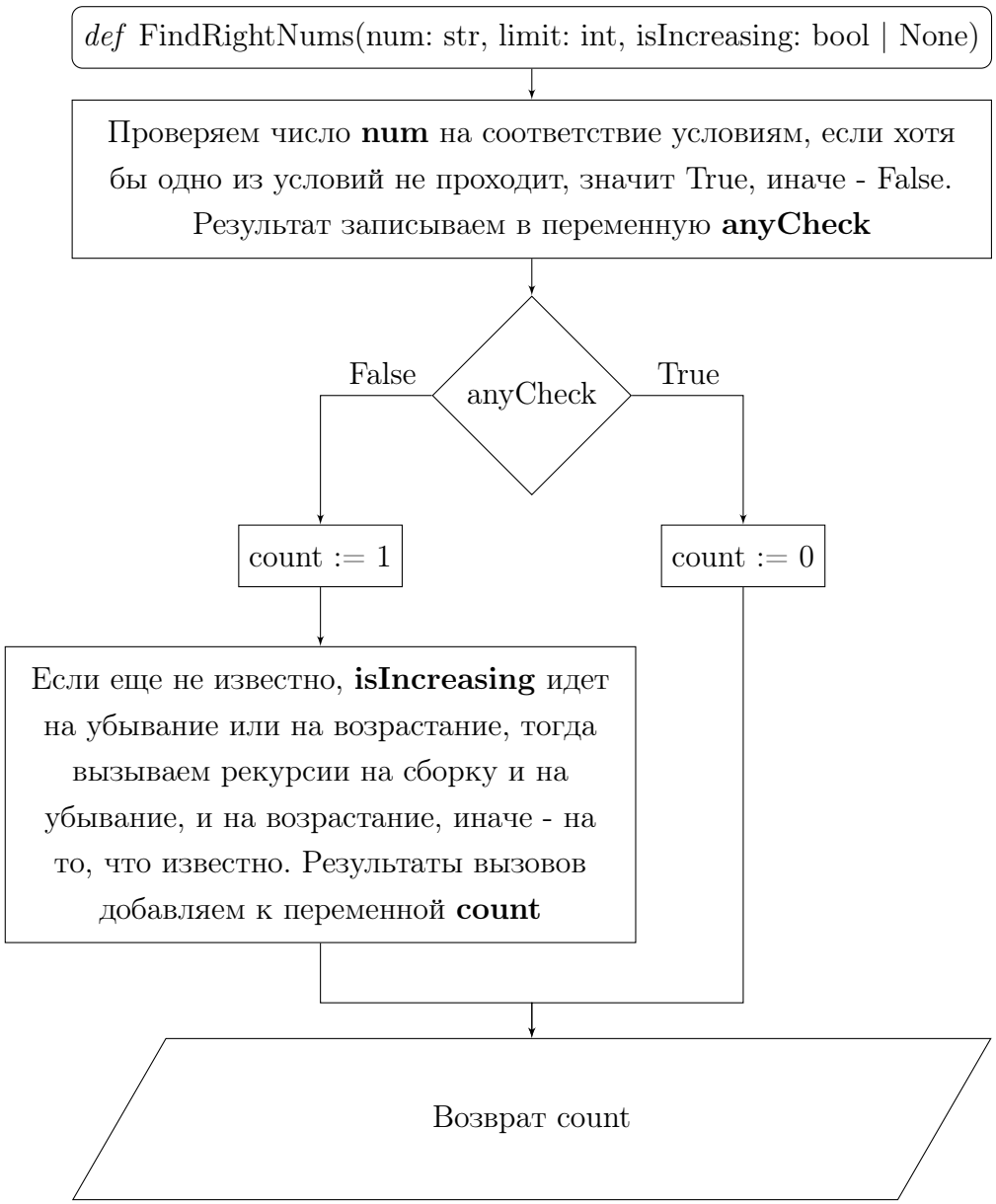
3. Спецификация функции *SearchNaturalNumsWithSortedDigits*:

   (a) Заголовок: `def SearchNaturalNumsWithSortedDigits(limit: int) -> int`

   (b) Назначение: используется для нахождения всех натуральных чисел, не превосходящих n, десятичная запись которых - строго возрастающая или строго убывающая последовательность цифр.

Блок-схемы:


def SearchNaturalNumsWithSortedDigits(limit: int)
countRightNums := 0
startNum := 1 . . . 10
Конец цикла
Вызовем FindRightNums(str(startNum), limit, None), передав startNum как строку, и добавим возвращенный результат к countRightNums
Возврат countRightNums
Конец


def FindRightNums(num: str, limit: int, isIncreasing: bool | None)
Проверяем число **num** на соответствие условиям, если хотя бы одно из условий не проходит, значит True, иначе - False. Результат записываем в переменную **anyCheck**
anyCheck
False
True
count := 1
count := 0
Если еще не известно, **isIncreasing** идет на убывание или на возрастание, тогда вызываем рекурсии на сборку и на убывание, и на возрастание, иначе - на то, что известно. Результаты вызовов добавляем к переменной **count**
Возврат count

4. Код *алгоритма* на языке *Python*:

```python
def SearchNaturalNumsWithSortedDigits(limit: int) -> int:
    countRightNums: int = 0
    for startNum in range(1, 10):
        countRightNums += FindRightNums(str(startNum), limit, None)

    return countRightNums


def FindRightNums(num: str, limit: int, isIncreasing: bool | None) -> int:
    numGreaterLimit: bool = (int(num) > limit)
    numIsStrictlyIncreasing: bool = (
        len(num) > 1 and int(num[-1]) <= int(num[-2])) if isIncreasing \
        else len(num) > 1 and int(num[-1]) >= int(num[-2])
    numIsBig: bool = len(num) > 10
    anyCheck: bool = numGreaterLimit or numIsStrictlyIncreasing or numIsBig

    if anyCheck:
        count: int = 0

    else:
        count: int = 1
        if isIncreasing or isIncreasing is None:
            for nextDigit in range(int(num[-1]), 10):
                count += FindRightNums(num + str(nextDigit), limit, True)
        if not isIncreasing or isIncreasing is None:
            for nextDigit in range(0, int(num[-1])):
                count += FindRightNums(num + str(nextDigit), limit, False)

    return count
```

5. *Таблица 7*

Тестовые данные

Входные данные	Выходные данные
50	46
100	90
500	174
1000	294
10000000	1458

2.8 Тема 8. Геометрия и теория множеств. Задание 23.

1. Медианой множества точек на плоскости назовём прямую, которая делит множество на два подмножества одинаковой мощности. Найти горизонтальную и вертикальную медианы заданного множества, у которого никакие две точки не лежат на одной горизонтальной или вертикальной прямой.

2. Словесное описание *алгоритма*:

   (a) Для решения задачи необходимо найти горизонтальную и вертикальную медианы, соответствующие условию. Для этого отсортируем точки сначала по x, затем - по y, взяв медианы из отсортированных точек.

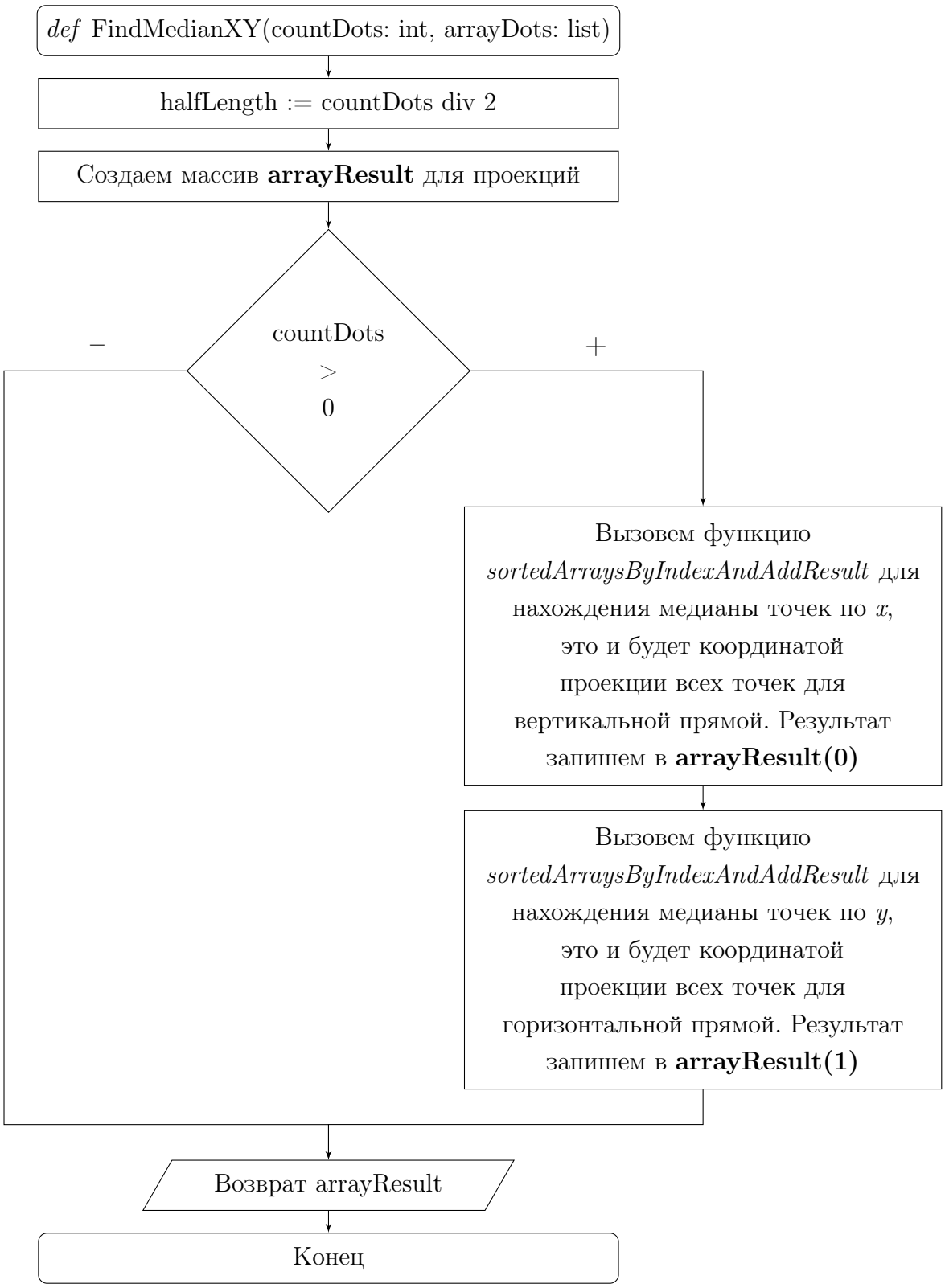
   (b) Полученные медианы будут являться координатами проекции всех точек для построения медиан по условию.

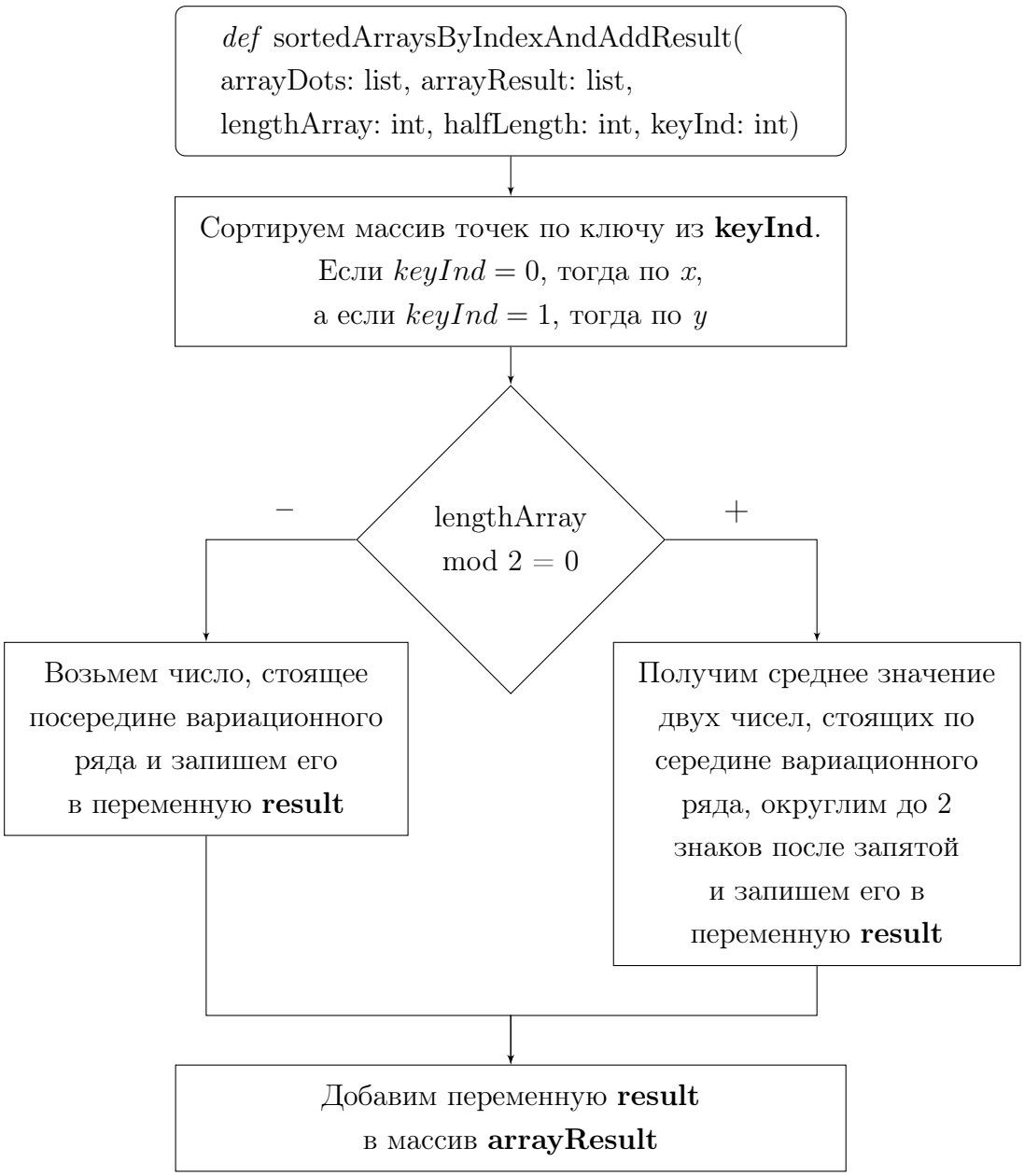
3. Спецификация функции *FindMedianXY*:

   (a) Заголовок: `def FindMedianXY(countDots: int, arrayDots: list) -> list`

   (b) Назначение: используется для нахождения координат проекции всех точек прямой, являющихся медианами, которые делят вертикально и горизонтально множество точек на два равных по мощности подмножества.

Блок-схемы:


def FindMedianXY(countDots: int, arrayDots: list)
halfLength := countDots div 2
Создаем массив arrayResult для проекций
countDots > 0
−
+
Вызовем функцию sortedArraysByIndexAndAddResult для нахождения медианы точек по x, это и будет координатой проекции всех точек для вертикальной прямой. Результат запишем в arrayResult(0)
Вызовем функцию sortedArraysByIndexAndAddResult для нахождения медианы точек по y, это и будет координатой проекции всех точек для горизонтальной прямой. Результат запишем в arrayResult(1)
Возврат arrayResult
Конец


def sortedArraysByIndexAndAddResult(
arrayDots: list, arrayResult: list,
lengthArray: int, halfLength: int, keyInd: int)
Сортируем массив точек по ключу из keyInd.
Если keyInd = 0, тогда по x,
а если keyInd = 1, тогда по y
lengthArray
mod 2 = 0
−
+
Возьмем число, стоящее
посередине вариационного
ряда и запишем его
в переменную result
Получим среднее значение
двух чисел, стоящих по
середине вариационного
ряда, округлим до 2
знаков после запятой
и запишем его в
переменную result
Добавим переменную result
в массив arrayResult

4. Код *алгоритма* на языке *Python*:

```python
def FindMedianXY(countDots: int, arrayDots: list) -> list:
    halfLength: int = countDots // 2
    arrayResult: list = []

    if countDots > 0:
        # вертикальная
        sortedArraysByIndexAndAddResult(arrayDots, arrayResult, countDots,
                                        halfLength, 0)

        # горизонтальная
        sortedArraysByIndexAndAddResult(arrayDots, arrayResult, countDots,
                                        halfLength, 1)

    return arrayResult


def sortedArraysByIndexAndAddResult(arrayDots: list, arrayResult: list,
                                    lengthArray: int, halfLength: int,
                                    keyInd: int) -> None:
    arrayDots.sort(key=lambda x: x[keyInd])
    if lengthArray % 2 == 0:
        result = round((arrayDots[halfLength][keyInd] +
                        arrayDots[halfLength - 1][keyInd]) / 2, 2)
    else:
        result = arrayDots[halfLength][keyInd]

    arrayResult.append(result)
```

5. *Таблица 8*

Тестовые данные

Входные данные	Выходные данные
4 3 4 1 2 7 8 5 6	4.00 5.00
5 4 6 1 7 10 20 11 13 18 26	10 13
1 10 20	10 20
4 4.5 5.4 1.1 7.2 11.3 5.4 12.3 7.7	7.90 6.30

2.9 Тема 9. Линейная алгебра и сжатие информации. Задание 23.

1. Заданный неупакованный двоичный массив сжать, используя полубайтовое представление длин цепочек.

2. Словесное описание *алгоритма*:

   (a) Задача довольно непростая, так как подобные способы сжатия практически не используются. Разработаем для этого свой алгоритм.

   (b) Будем отталкиваться, что полбайта - это 4 бит. Максимальное число - 15.

   (c) Считаем, что массив начинается с 1, то есть нет незначащих нулей.

   (d) Будем хранить в сжатом виде лишь количество подряд идущих бит, то есть цепочки.

   (e) Придерживаться будем следующих правил:

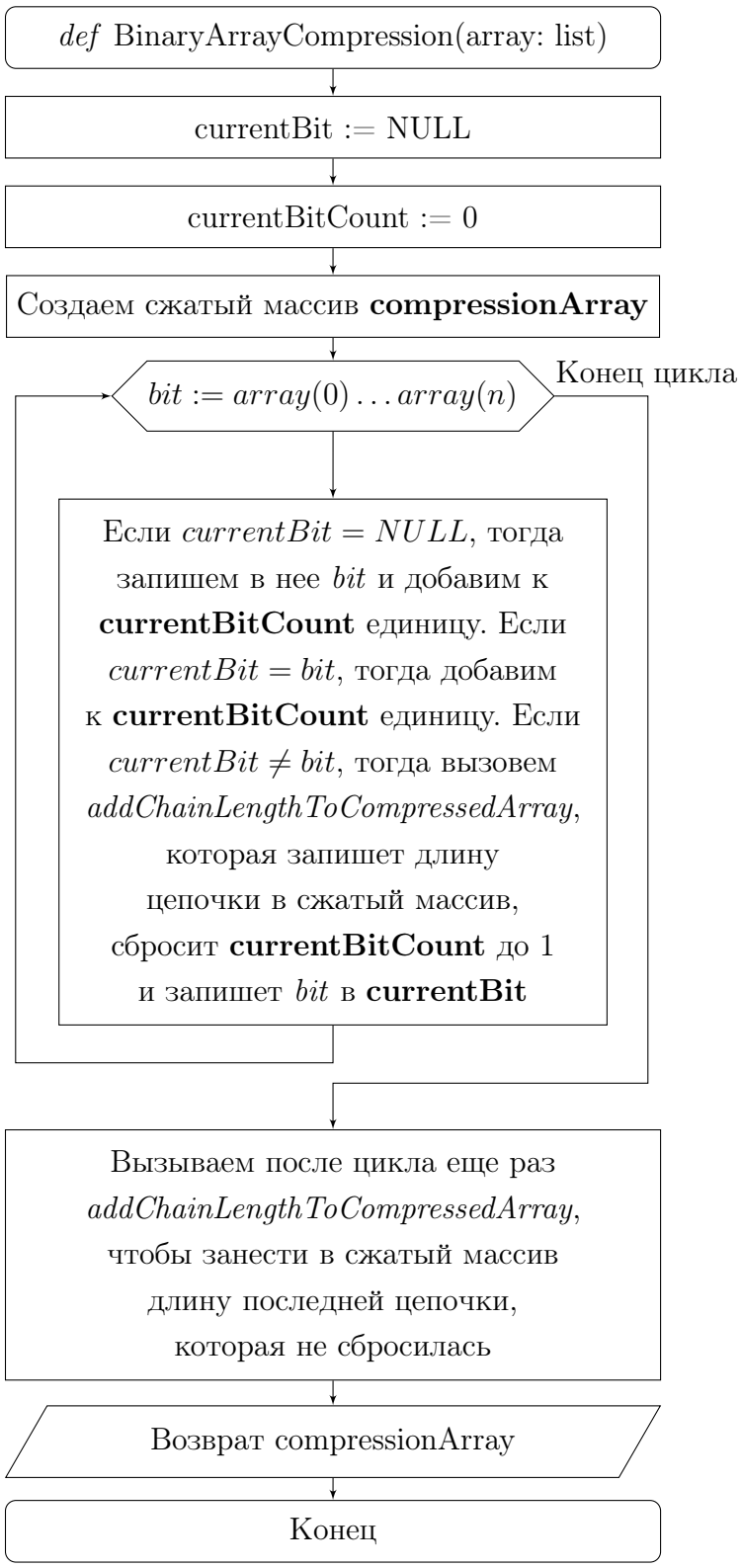
      1. Если длина цепочки меньше 15, тогда записываем фактическую длину цепочки.
      2. Если длина цепочки равна 15, тогда запишем в сжатый массив два числа - 15 и 0. 0 будет признаком того, что длина цепочки кратна 15. Дальше может идти лишь противоположный бит исходного массива.
      3. Если длина цепочки больше 15, тогда делаем переносы, записывая длину числами, не превышающими 15, до полного разложения исходной длины. Если после какого-то числа 15 идет не 0, значит это число также относится к цепочке. На примере: имеем 31 подряд идущий бит 0 в исходном массиве, а затем 3 подряд идущие 1. Запишем это так: [15, 15, 1, 3]. А если в исходном массиве будет 30 подряд идущих бит 0, а затем 3 подряд идущие 1, тогда запишем это так: [15, 15, 0, 3].

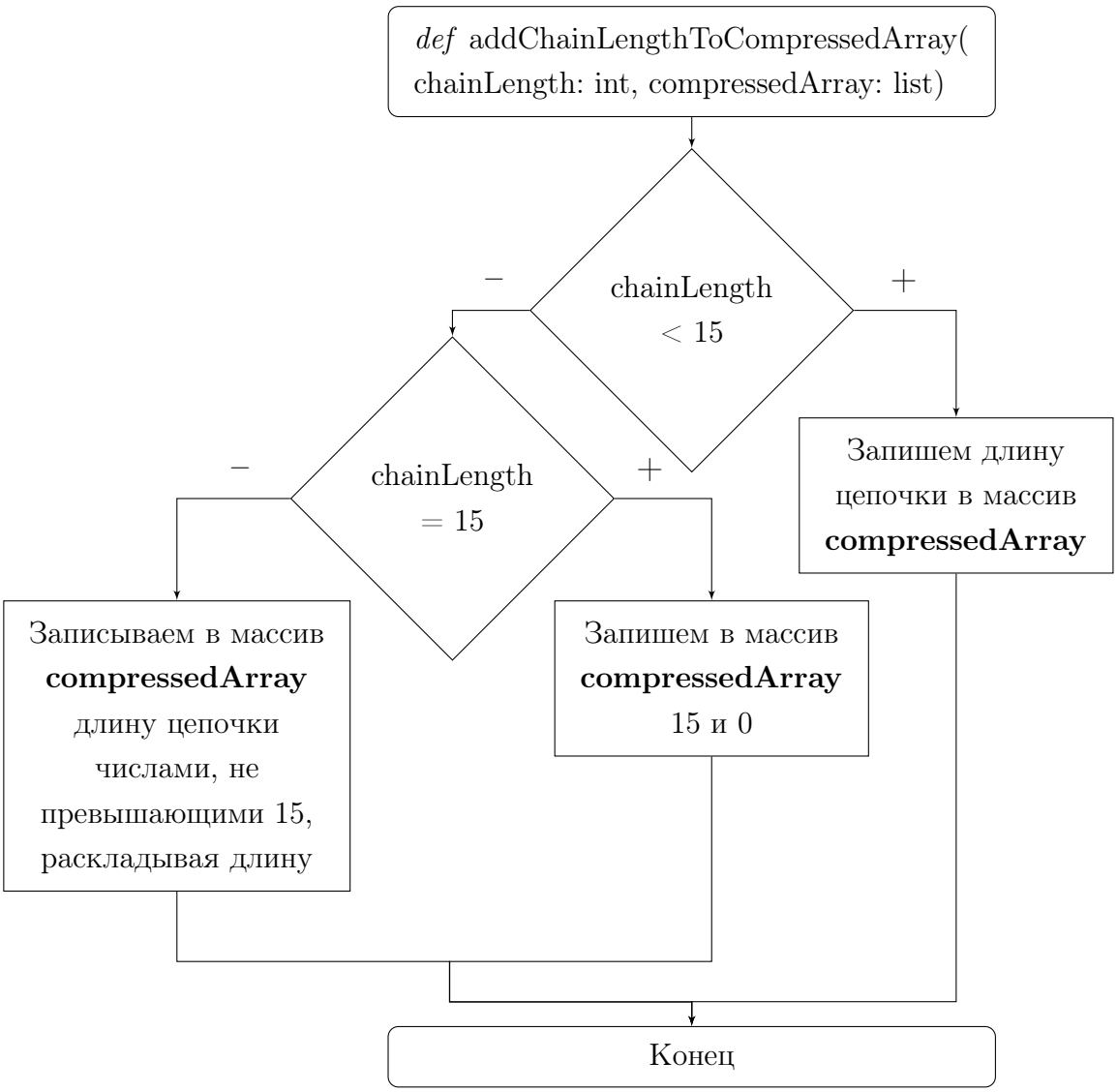
3. Спецификация функции *BinaryArrayCompression*:

   (a) Заголовок: `def BinaryArrayCompression(array: list) -> list`

   (b) Назначение: используется для сжатия исходного неупакованного двоичного массива, применяя способ полубайтового представления длин цепочек.

Блок-схемы:


def BinaryArrayCompression(array: list)
currentBit := NULL
currentBitCount := 0
Создаем сжатый массив compressionArray
bit := array(0) . . . array(n)
Конец цикла
Если currentBit = NULL, тогда запишем в нее bit и добавим к currentBitCount единицу. Если currentBit = bit, тогда добавим к currentBitCount единицу. Если currentBit ≠ bit, тогда вызовем addChainLengthToCompressedArray, которая запишет длину цепочки в сжатый массив, сбросит currentBitCount до 1 и запишет bit в currentBit
Вызываем после цикла еще раз addChainLengthToCompressedArray, чтобы занести в сжатый массив длину последней цепочки, которая не сбросилась
Возврат compressionArray
Конец


def addChainLengthToCompressedArray(chainLength: int, compressedArray: list)
chainLength < 15
−
+
chainLength = 15
−
+
Записываем в массив compressedArray длину цепочки числами, не превышающими 15, раскладывая длину
Запишем в массив compressedArray 15 и 0
Запишем длину цепочки в массив compressedArray
Конец

4. Код *алгоритма* на языке *Python*:

```python
def BinaryArrayCompression(array: list) -> list:
    currentBit: int | None = None
    currentBitCount: int = 0
    compressionArray: list = []

    for bit in array:
        if currentBit is None:
            currentBit = bit
            currentBitCount += 1
        elif bit == currentBit:
            currentBitCount += 1
        else:
            addChainLengthToCompressedArray(currentBitCount, compressionArray)
            currentBitCount = 1
            currentBit = bit

    if currentBitCount > 0:
        addChainLengthToCompressedArray(currentBitCount, compressionArray)

    return compressionArray


def addChainLengthToCompressedArray(chainLength: int,
                                    compressedArray: list) -> None:
    if chainLength < 15:
        compressedArray.append(chainLength)
    elif chainLength == 15:
        compressedArray.append(chainLength)
        compressedArray.append(0)
    else:
        while chainLength > 0:
            if chainLength >= 15:
                numAdd = 15
            else:
                numAdd = chainLength

            chainLength -= numAdd
            compressedArray.append(numAdd)
```

5. *Таблица 9*

Тестовые данные

Входные данные	Выходные данные
1 1 1 0 0 1	3 2 1
1 0 1 0 1 0	1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0	15 0 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1	15 2 1 1