**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)**

**Институт информационных технологий и**

**электроники**

**Кафедра информации и защиты информации**

**ПРАКТИЧЕСКАЯ РАБОТА № 2**

по дисциплине «Технологии и методы программирования»

**«ФУНКЦИИ. РЕКУРСИВНЫЕ ФУНКЦИИ»**

**Выполнил:**

Масляных Анастасия Дмитриевна ИСБ-120

**Принял:**

Доцент кафедры ИЗИ

Артюшина Лариса Андреевна

Владимир 2024

**СОДЕРЖАНИЕ**

[ПРАКТИЧЕСКОЕ ЗАДАНИЕ 3](#_Toc159171541)

[БЛОК-СХЕМА АЛГОРИТМА 4](#_Toc159171542)

[ТЕСТИРОВАНИЕ 6](#_Toc159171543)

[ТЕКСТ ПРОГРАММЫ 7](#_Toc159171544)

[ВЫВОД 11](#_Toc159171545)

# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Вариант №6  
  
Текст задания: Получить N троек чисел Пифагора (натуральные числа a, b и c называются числами Пифагора, если выполняется условие ).

# БЛОК-СХЕМА АЛГОРИТМА

|  |  |
| --- | --- |
| А) | Б) |

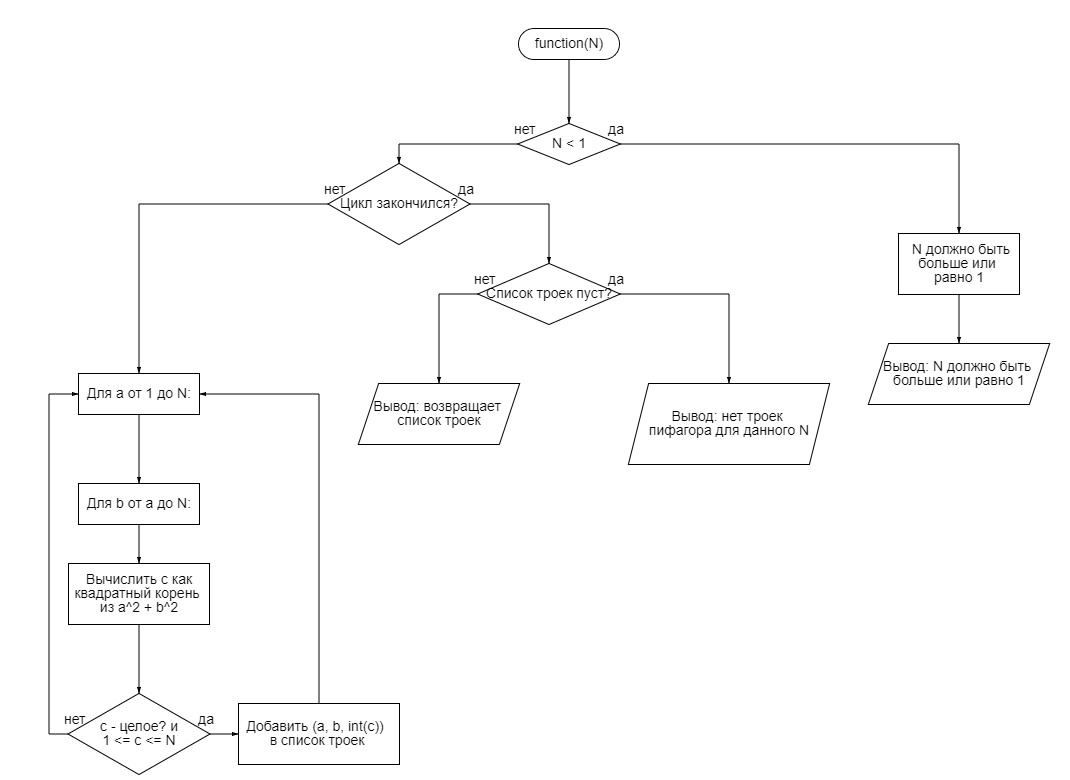
Блок-схема основной функции:В) 

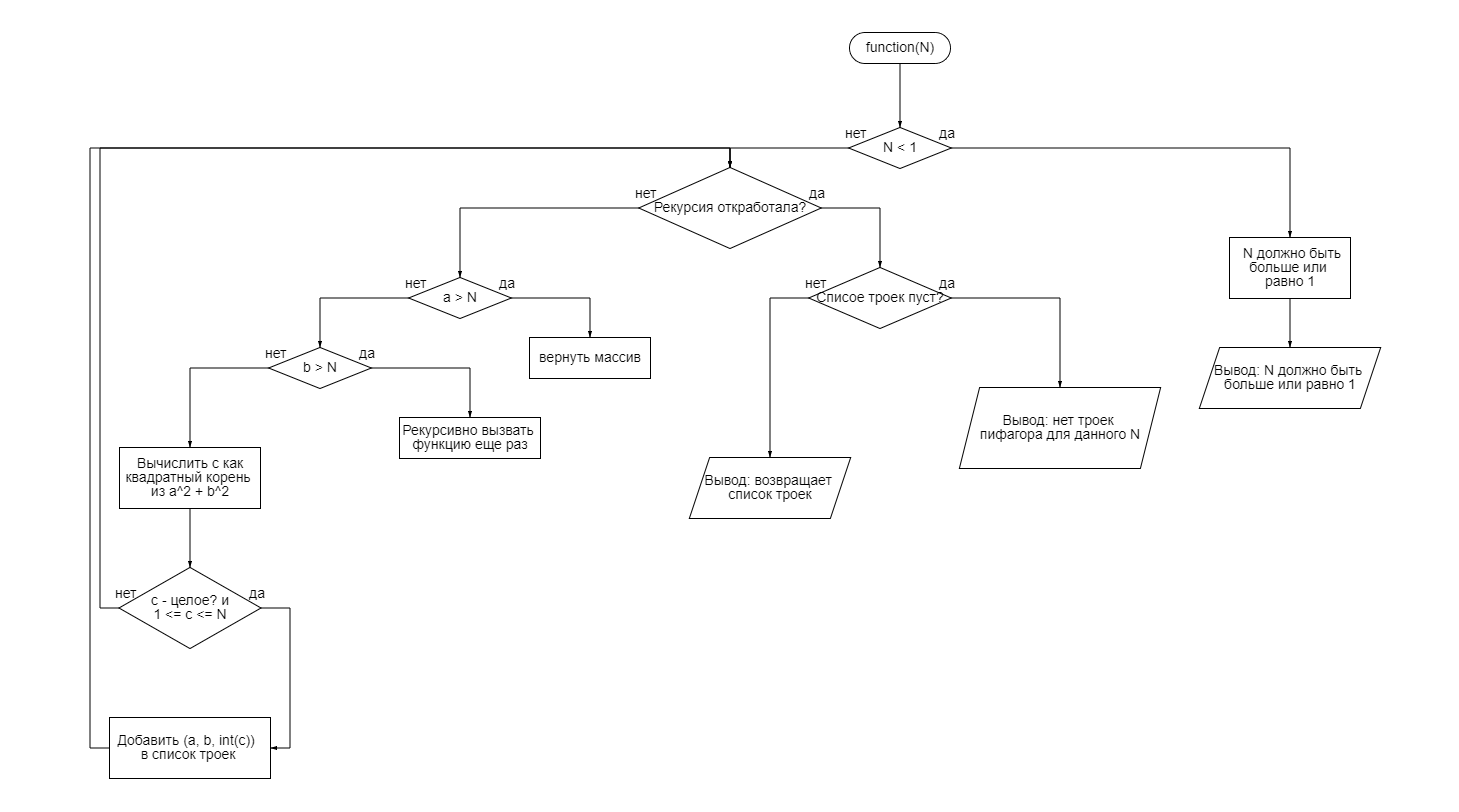
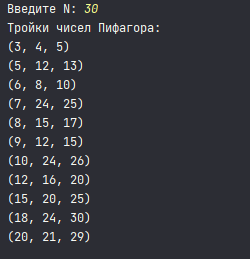
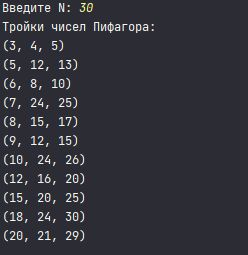
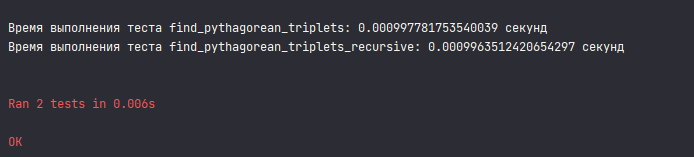
Рис. 1 АБВ Алгоритм задачи для итеративного метода  
Г)

Рис. 1 АБГ Алгоритм задачи для рекурсивного метода

# ТЕСТИРОВАНИЕ

Вывод итеративного метода:  
  
  
  
Вывод рекурсивного метода:  
  
  
Юнит-тесты:  
  


# ТЕКСТ ПРОГРАММЫ

Текст программы lb2.py:

import math

class Pythagorean:  
  
 def find\_pythagorean\_triplets(*self*, *N*: *int*):  
 *"""  
 Функция для нахождения троек чисел Пифагора.  
  
 Args:  
 - N: int, верхняя граница для поиска троек чисел Пифагора.  
  
 Returns:  
 - list: Список троек чисел Пифагора, удовлетворяющих условиям.  
 """* if *N* < 1:  
 raise *ValueError*("N должно быть больше или равно 1")  
  
 triplets = []  
 for a in *range*(1, *N*+1):  
 for b in *range*(a, *N*+1):  
 c = math.sqrt(a\*\*2 + b\*\*2)  
 if c.is\_integer() and 1 <= c <= *N*:  
 triplets.append((a, b, *int*(c)))  
  
 if not triplets:  
 raise *ValueError*("Нет троек пифагора для данного N")  
 return triplets  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 Pythagorean = Pythagorean()  
 N = *int*(*input*("Введите N: "))  
 pythagorean\_triplets = Pythagorean.find\_pythagorean\_triplets(N=N)  
 *print*("Тройки чисел Пифагора:")  
 for triplet in pythagorean\_triplets:  
 *print*(triplet)  
 except *ValueError* as ve:  
 *print*(f"Ошибка: {ve}")

Текст программы lb2\_1.py:

import math  
  
class Pythagorean\_recursive:  
  
 def find\_pythagorean\_triplets\_recursive(*self*, *N*: *int*, *a*=1, *b*=1, *triplets*=None):  
 *"""  
 Функция для нахождения троек чисел Пифагора с помощью рекурсии.  
  
 Args:  
 - N: int, верхняя граница для поиска троек чисел Пифагора.  
 - a: int, текущее значение a.  
 - b: int, текущее значение b.  
 - triplets: list, текущий список троек чисел Пифагора.  
  
 Returns:  
 - list: Список троек чисел Пифагора, удовлетворяющих условиям.  
 """* if *triplets* is None:  
 triplets = []  
  
 if *N* < 1:  
 raise *ValueError*("N должно быть больше или равно 1")  
  
 if *a* > *N*:  
 return *triplets* if *b* > *N*:  
 return *self*.find\_pythagorean\_triplets\_recursive(*N*, *a* + 1, *a* + 1, *triplets*)  
  
 c = math.sqrt(*a* \*\* 2 + *b* \*\* 2)  
 if c.is\_integer() and 1 <= c <= *N*:  
 *triplets*.append((*a*, *b*, *int*(c)))  
 result = *self*.find\_pythagorean\_triplets\_recursive(*N*, *a*, *b* + 1, *triplets*)  
 if not result:  
 raise *ValueError*("Нет троек пифагора для данного N")  
 return result  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 Pythagorean = Pythagorean\_recursive()  
 N = *int*(*input*("Введите N: "))  
 pythagorean\_triplets = Pythagorean.find\_pythagorean\_triplets\_recursive(N=N)  
 *print*("Тройки чисел Пифагора:")  
 for triplet in pythagorean\_triplets:  
 *print*(triplet)  
 except *ValueError* as ve:  
 *print*(f"Ошибка: {ve}")

Текст юнит-тестов программы:

import unittest  
import time  
from lab2 import Pythagorean  
from lab2\_1 import Pythagorean\_recursive  
  
Pythagorean = Pythagorean()  
Pythagorean\_recursive = Pythagorean\_recursive()  
  
class TestPythagoreanTriplets(unittest.TestCase):  
  
 def test\_find\_pythagorean\_triplets(*self*):  
 start\_time = time.time()  
 *# Проверяем, что функция возвращает ожидаемый результат для N = 5* expected\_result\_N5 = [(3, 4, 5)]  
 *self*.assertEqual(Pythagorean.find\_pythagorean\_triplets\_iterative(N=5), expected\_result\_N5)  
  
 *# Проверяем, что функция возвращает ожидаемый результат для N = 10* expected\_result\_N10 = [(3, 4, 5), (6, 8, 10)]  
 *self*.assertEqual(Pythagorean.find\_pythagorean\_triplets\_iterative(N=10), expected\_result\_N10)  
  
 expected\_result\_N40 = [  
 (3, 4, 5),  
 (5, 12, 13),  
 (6, 8, 10),  
 (7, 24, 25),  
 (8, 15, 17),  
 (9, 12, 15),  
 (10, 24, 26),  
 (12, 16, 20),  
 (12, 35, 37),  
 (15, 20, 25),  
 (15, 36, 39),  
 (16, 30, 34),  
 (18, 24, 30),  
 (20, 21, 29),  
 (21, 28, 35),  
 (24, 32, 40),  
 ]  
 *self*.assertEqual(Pythagorean.find\_pythagorean\_triplets\_iterative(40), expected\_result\_N40)  
  
 *# Проверяем, что функция обрабатывает ошибку при N = 2 (нет троек)* with *self*.assertRaises(*ValueError*):  
 Pythagorean.find\_pythagorean\_triplets\_iterative(N=2)  
  
 *# Проверяем, что функция обрабатывает ошибку при N = 0* with *self*.assertRaises(*ValueError*):  
 Pythagorean.find\_pythagorean\_triplets\_iterative(N=0)  
  
 *# Проверяем, что функция обрабатывает ошибку при N < 0* with *self*.assertRaises(*ValueError*):  
 Pythagorean.find\_pythagorean\_triplets\_iterative(N=-5)  
 end\_time = time.time()  
 *print*(f"Время выполнения теста find\_pythagorean\_triplets: {end\_time - start\_time} секунд")  
  
 *# @unittest.skip("Время выполнения теста find\_pythagorean\_triplets\_recursive слишком велико")* def test\_find\_pythagorean\_triplets\_recursive(*self*):  
 start\_time = time.time()  
 *# Тестирование для N = 5* expected\_result\_N5 = [(3, 4, 5)]  
 *self*.assertEqual(Pythagorean\_recursive.find\_pythagorean\_triplets\_recursive(5), expected\_result\_N5)  
  
 *# Тестирование для N = 10* expected\_result\_N10 = [(3, 4, 5), (6, 8, 10)]  
 *self*.assertEqual(Pythagorean\_recursive.find\_pythagorean\_triplets\_recursive(10), expected\_result\_N10)  
  
 expected\_result\_N40 = [  
 (3, 4, 5),  
 (5, 12, 13),  
 (6, 8, 10),  
 (7, 24, 25),  
 (8, 15, 17),  
 (9, 12, 15),  
 (10, 24, 26),  
 (12, 16, 20),  
 (12, 35, 37),  
 (15, 20, 25),  
 (15, 36, 39),  
 (16, 30, 34),  
 (18, 24, 30),  
 (20, 21, 29),  
 (21, 28, 35),  
 (24, 32, 40),  
 ]  
 *self*.assertEqual(Pythagorean\_recursive.find\_pythagorean\_triplets\_recursive(40), expected\_result\_N40)  
  
 *# Тестирование для N = 2 (нет троек)* with *self*.assertRaises(*ValueError*):  
 Pythagorean.find\_pythagorean\_triplets\_iterative(N=2)  
  
 *# Тестирование для N = 0 (вызов исключения)* with *self*.assertRaises(*ValueError*):  
 Pythagorean\_recursive.find\_pythagorean\_triplets\_recursive(2)  
  
 *# Тестирование для N < 0 (вызов исключения)* with *self*.assertRaises(*ValueError*):  
 Pythagorean\_recursive.find\_pythagorean\_triplets\_recursive(-5)  
 end\_time = time.time()  
 *print*(f"Время выполнения теста find\_pythagorean\_triplets\_recursive: {end\_time - start\_time} секунд")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

# ВЫВОД

Итеративная реализация выполняется быстрее, чем рекурсивная реализация.

В случае итеративной реализации алгоритма время выполнения программы линейно зависит от N.

В случае рекурсивной реализации алгоритма время выполнения программы зависит от глубины рекурсии, которая также зависит от N. В результате рекурсивная реализация работает медленнее, чем итеративная реализация.