**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Алгоритмы и структуры данных»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

«Сортировка методом Монте-Карло»

**Выполнил:**

Гачко Г. Д., студент группы N3246

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Ерофеев С. А.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Санкт-Петербург

2024 г.

Содержание

[Введение 4](#_Toc159025670)

[1 Описание функционала программы 5](#_Toc159025671)

[2 Блок-схема 6](#_Toc159025672)

[3 Код программы 7](#_Toc159025673)

[4 Результаты тестирования 8](#_Toc159025674)

[Заключение 10](#_Toc159025675)

Введение

Задача работы – разработать программу сортировки массива чисел из файла методом Монте-Карло.

Для выполнения задачи требуется реализовать ввод чисел из файла, выполнение сортировки и вывод чисел в файл.

Для реализации был выбран язык программирования Python версии 3.10.4.

# Описание функционала программы

Программа принимает на вход из stdin режим запуска – ручное управление (M) или запуск тестов (T). Программа обращается к файлу с входными данными по жестко указанному в коде относительному пути к файлу. Это решение было принято из соображений простоты использования программы. Далее будут рассмотрены два режима работы программы.

**M**anual:

В этом режиме программы в бесконечном цикле выполняет сортировку файла numbers\_valid.txt. Как только сортировка завершена, программа выводит в stdout результирующий массив и ожидает подтверждения следующей сортировки путем нажатия клавиши Enter. Между сортировками входной файл numbers\_valid.txt может быть изменен пользователем для проверки функционирования программы при различных входных данных.

**T**est:

В этом режиме программа не принимает пользовательский ввод, а запускается на наборе заранее написанных тестов. Встроенные в программу тесты включают проверку следующих случаев:

1. Во входном файле содержатся данные, не приводимые к типу float (буквы, знаки препинания, пробельные символы и т.п.)
2. Во входном файле содержатся числа, превышающие размер типа данных float для языка Python 3.11
3. Во входном файле содержатся корректные данные: отрицательные и положительные дробные и целые числа, ноль.

В обоих режимах работы алгоритм программы выглядит следующим образом. При получении данных из файла программа проводит проверку данных на приводимость к типу данных float, и завершается с ошибкой, если данные из файла невозможно преобразовать к типу float. Также проводится проверка на переполнение типа данных float.

После запуска программа считывает массив чисел из входного файла и закрывает его. Далее проводится случайная перестановка чисел в массиве до тех пор, пока массив не будет отсортирован. После выполнения сортировки программа записывает отсортированный массив в файл, отличный от файла ввода, и выводит получившийся массив в stdout.

# Используемые переменные

filenameForTest1, filenameForTest2, filenameForTest3, filenameOutput – имена файлов для работы программы (тип string, диапазон ограничен памятью устройства)

numbers – массив сортируемых чисел (тип float List, максимальная длина – 9223372036854775807 элементов)

theGoal – массив для проверки успешности сортировки (тип float List, максимальная длина – 9223372036854775807 элементов)

content – переменная для промежуточного хранения данных во время чтения из файла (тип string, диапазон ограничен памятью устройства)

D – дискриминант квадратного уравнения (тип float)

x, x1, x2 – корни уравнения (тип float)

# Блок-схема

# Код программы

import math

def tetra\_solve(a = 'error', b = 'error', c = 'error', testing\_mode = False):

    if not testing\_mode:

        try:

            # Ввод переменных.

            # float a - коэффициент при x^2

            # float b - коэффициент при x^1

            # float c - коэффициент при x^0

            a, b, c = map(float, input("Введите коэффициенты квадратного уравнения, начиная с коэффициента при x^2 (a b c):\n>>> ").split())

        except ValueError:

            print("Введите число, будьте человеком") if not testing\_mode else False

            return False

        if (a == a + 1) or (b == b + 1) or (c == c + 1):

            print("Один из коэффициентов выходит за рамки допустимых значений типа float") if not testing\_mode else False

            return False

    if (a == 0):

        if (b == 0):

            if (c == 0):

                # Случай a == 0 И b == 0 И с == 0.

                # Если все коэффициенты == 0, то уравнение обращается в

                # 0 = 0

                # Следовательно, уравнение верно при любых x

                print("x - любое") if not testing\_mode else False

                return True

            else:

                # Случай a == 0 И b == 0 И с != 0.

                # Если коэффициенты при x == 0, а c != 0, то уравнение обращается в

                # с = 0, где с != 0

                # Следовательно, уравнение ложно при любых x

                print("нет корней") if not testing\_mode else False

                return {}

        else:

            # Случай a == 0 И b != 0 И с - любое

            # Если коэффициент при x^2 == 0, а b != 0, то уравнение обращается в линейное

            # bx + c = 0, где b != 0

            # Следовательно, решим линейное уравнение с помощью формулы

            # x =  -c / b

            print("уравнение обращается в линейное") if not testing\_mode else False

            x = -c / b

            print("%.4f" % x) if not testing\_mode else False

            return {x}

    else:

        # Случай a != 0 И b,c - любые

        # Если коэффициент при x^2 != 0, то уравнение является квадратным.

        # Определим количество и вид его корней через формулу дискриминанта

        # D = b\*b + (-4) \* a \* c

        D = (b\*b) + ((-4) \* a \* c) # float D - дискриминант квадратного уравнения

        if D > 0:

            # Случай D > 0

            # Если дискриминант больше нуля, то квадратное уравнение имеет два различных действительных корня

            # Обозначим их как x1 и x2

            # И найдем по стандартным формулам для корней квадратного уравнения, представленным ниже

            x1 = ( -b + math.sqrt(D) ) / (2\*a) # float x1 - первый корень квадратного уравнения

            x2 = ( -b - math.sqrt(D) ) / (2\*a) # float x2 - второй корень квадратного уравнения

            print("D > 0") if not testing\_mode else False

            print("%.4f" % x1) if not testing\_mode else False

            print("%.4f" % x2) if not testing\_mode else False

            return {x1, x2}

        elif D == 0:

            # Случай D == 0

            # Если дискриминант равен нулю, то квадратное уравнение имеет два кратных (одинаковых) действительных корня

            # Обозначим их как x1 = x2 = x

            # И найдем по стандартной формуле для кратных корней квадратного уравнения, представленной ниже

            #print("D == 0") if not testing\_mode else False

            x = (-b) / (2\*a)

            print("%.4f" % x) if not testing\_mode else False

            return {x}

        elif D < 0:

            # Случай D < 0

            # Если дискриминант меньше нуля, то квадратное уравнение имеет два мнимых корня

            # В рамках нашего алгоритма мы не занимаемся их вычислением

            # И найдем по стандартной формуле для кратных корней квадратного уравнения, представленной ниже

            #print('a > 10') if not testing\_mode else False

            print("нет действительных корней") if not testing\_mode else False

            return {}

mode = input("Какой режим запуска? (run [T]ests / run [M]anually)")

if mode == 'M':

    while True:

    # функция вычисления корня будет автоматически запускаться заново после конца текущего вызова

        tetra\_solve()

elif mode == 'T':

    # Шесть тестов

    # Случай D > 0

    # Если дискриминант больше нуля, то квадратное уравнение имеет два различных действительных корня

    # Обозначим их как x1 и x2

    # И найдем по стандартным формулам для корней квадратного уравнения, представленным ниже

    assert {} == tetra\_solve(3, -4, 94, testing\_mode = True)

    print("Тест D > 0 успешен")

    # Случай D == 0

    # Если дискриминант равен нулю, то квадратное уравнение имеет два кратных (одинаковых) действительных корня

    # Обозначим их как x1 = x2 = x

    # И найдем по стандартной формуле для кратных корней квадратного уравнения, представленной ниже

    assert {3.5} == tetra\_solve(-4, 28, -49, testing\_mode = True)

    print("Тест D == 0 успешен")

    # Случай D < 0

    # Если дискриминант меньше нуля, то квадратное уравнение имеет два мнимых корня

    # В рамках нашего алгоритма мы не занимаемся их вычислением

    # И найдем по стандартной формуле для кратных корней квадратного уравнения, представленной ниже

    assert {3, -3} == tetra\_solve(-6, 0, 54, testing\_mode = True)

    print("Тест D < 0 успешен")

    # Случай a == 0 И b == 0 И с == 0.

    # Если все коэффициенты == 0, то уравнение обращается в

    # 0 = 0

    # Следовательно, уравнение верно при любых x

    assert True == tetra\_solve(0, 0, 0, testing\_mode = True)

    print("Тест a = 0, b = 0, c = 0 успешен")

    # Случай a == 0 И b == 0 И с != 0.

    # Если коэффициенты при x == 0, а c != 0, то уравнение обращается в

    # с = 0, где с != 0

    # Следовательно, уравнение ложно при любых x

    assert {} == tetra\_solve(0, 0, 4, testing\_mode = True)

    print("Тест a == 0, b == 0, с != 0 успешен")

    # Случай a == 0 И b != 0 И с - любое

    # Если коэффициент при x^2 == 0, а b != 0, то уравнение обращается в линейное

    # bx + c = 0, где b != 0

    # Следовательно, решим линейное уравнение с помощью формулы

    # x =  -c / b

    assert {-2} == tetra\_solve(0, 4, 8, testing\_mode = True)

    print("Тест линейного уравнения успешен")

else:

    print("Нет такого режима")

# Результаты тестирования

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описаниеБыли проведены тесты программы с различными входными данными. Все реализованные случаи ввода корректно обрабатываются и отображаются. Тестирование производилось с помощью интерпретатора Python 3.10.4 и среды разработки [ms-python.python](https://marketplace.visualstudio.com/items?itemName=ms-python.python) версии 2024.3.10791010

# Заключение

В ходе выполнения данной лабораторной работы была достигнута поставленная задача. А именно, был составлен и реализован алгоритм по решению квадратного уравнения в области действительных чисел.

В ходе решения данной задачи была реализована функция «tetra\_solve», принимающая на вход коэффициенты квадратного уравнения, а также проверяющая соответствие введённых данных коэффициентам квадратного уравнения и решающая его.

Было выполнено тестирование программы на языке Python версии 3.10.4 с помощью среды разработки [ms-python.python](https://marketplace.visualstudio.com/items?itemName=ms-python.python) версии 2024.3.10791010, в ходе которого все реализованные выводы оказались корректно работающими.