**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 № 2**

«Интегрирование методом Гаусса-Лежандра с использованием сортировки расчёской и структуры данных дек»

**Выполнил:**

Гачко Г. Д., студент группы N3246

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

(подпись)

**Проверил:**

Ерофеев С. А.

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

(подпись)

Санкт-Петербург

2024 г.

Содержание

1. Введение
2. Описание функционала программы
3. Используемые переменные
4. Блок-схема
5. Код программы
6. Результаты тестирования
7. Заключение

Введение

Задача работы – разработать программу интегрирования методом Гаусса, считывая пары координат из файла и сортируя их расчёской. Использовать дек.

Для выполнения задачи требуется реализовать ввод пар координат из файла, выполнение сортировки, и вычисление интеграла.

Для реализации был выбран язык программирования Python версии 3.13.0.

# Описание функционала программы

Программа принимает на вход из файла пары координат и записывает их в дек. Далее пары координат сортируются в деке расческой. После этого программа вычисляет интеграл по заданным координатам методом Гаусса-Лежандра, и выводит ответ в stdout.

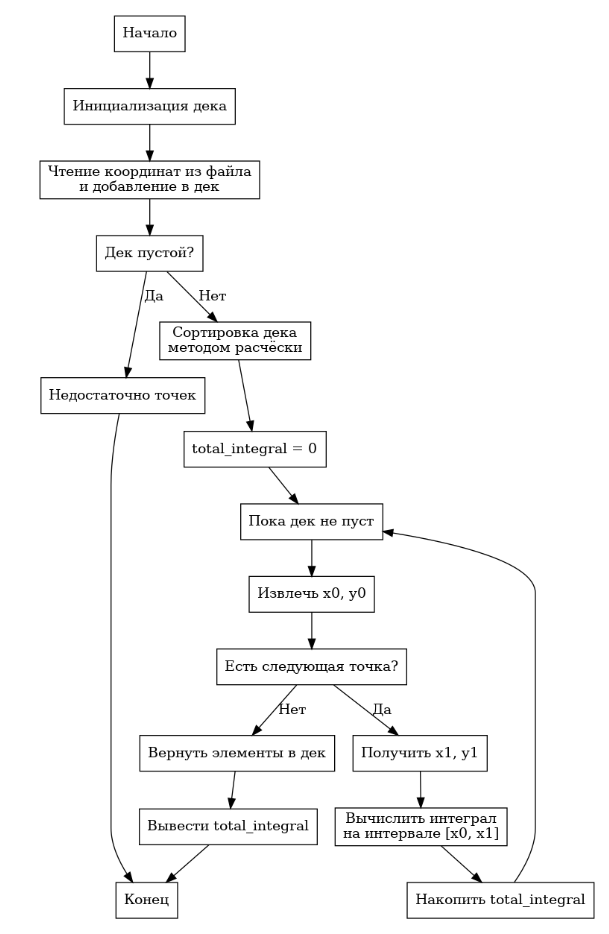
# Используемые переменные

coordinate\_pairs, temp\_deque, front\_deque, back\_deque – пары координат типа данных float, записанные в структуру данных типа дек, сделанную на нативной структуре данных языка Python – List (тип float List, максимальная длина – 9223372036854775807 элементов)

gap, size, shrink, first\_item, second\_item, x, y – разнообразные сервисные переменные типа float. Переменная типа float занимает 4 байта.

sorted – переменная типа bool.

# Блок-схема



# Код программы

import math

# Определяем класс Deque для реализации структуры данных дек

class Deque:

def \_\_init\_\_(self):

# Внутренний список для хранения элементов дека

self.items = []

def PushBack(self, item):

# Добавляем элемент в конец дека

self.items.append(item)

def PushFront(self, item):

# Добавляем элемент в начало дека

self.items.insert(0, item)

def PopBack(self):

# Удаляем и возвращаем элемент из конца дека

if self.IsEmpty():

raise IndexError("PopBack from empty Deque")

return self.items.pop()

def PopFront(self):

# Удаляем и возвращаем элемент из начала дека

if self.IsEmpty():

raise IndexError("PopFront from empty Deque")

return self.items.pop(0)

def IsEmpty(self):

# Проверяем, пуст ли дек

return len(self.items) == 0

def Clear(self):

# Очищаем дек

self.items = []

# Инициализируем дек для хранения пар координат

coordinate\_pairs = Deque()

# Открываем файл для чтения пар координат

with open('test.txt', 'r') as file:

# Проходим по каждой строке в файле

for line in file:

# Разделяем строку на значения x и y

try:

x\_str, y\_str = line.strip().split()

except:

print("Входные данные идут не парами")

exit()

# Преобразуем строки в числа с плавающей запятой

try:

x = float(x\_str)

y = float(y\_str)

except:

print("Входные данные не конвертируются в тип float")

exit()

# Добавляем пару координат в дек с помощью PushBack

coordinate\_pairs.PushBack((x, y))

def comb\_sort(deque):

# Инициализируем переменную для хранения длины дека

size = 0

# Копируем элементы дека в список для вычисления длины

temp\_deque = Deque()

while not deque.IsEmpty():

# Перемещаем элементы в временный дек и увеличиваем счетчик

temp\_deque.PushBack(deque.PopFront())

size += 1

# Возвращаем элементы обратно в исходный дек

while not temp\_deque.IsEmpty():

deque.PushBack(temp\_deque.PopFront())

# Инициализируем переменные для сортировки

gap = size # Размер шага

shrink = 1.3 # Коэффициент уменьшения шага

sorted = False # Флаг для отслеживания состояния сортировки

# Цикл продолжается, пока список не будет отсортирован

while not sorted:

# Обновляем значение шага

gap = int(gap / shrink)

if gap <= 1:

# Устанавливаем минимальный шаг равным 1

gap = 1

# Предполагаем, что это последний проход

sorted = True

# Инициализируем индексы для обхода дека

index = 0

while True:

# Инициализируем временные деки для доступа к элементам по индексам

front\_deque = Deque()

back\_deque = Deque()

# Перемещаем элементы из исходного дека в front\_deque до индекса index

for \_ in range(index):

if deque.IsEmpty():

break

front\_deque.PushBack(deque.PopFront())

# Получаем первый элемент для сравнения

if deque.IsEmpty():

break

first\_item = deque.PopFront()

# Перемещаем элементы в back\_deque до элемента с индексом index + gap

for \_ in range(gap - 1):

if deque.IsEmpty():

break

back\_deque.PushBack(deque.PopFront())

# Проверяем, есть ли второй элемент для сравнения

if deque.IsEmpty():

# Возвращаем все элементы обратно

front\_deque.PushBack(first\_item)

while not back\_deque.IsEmpty():

front\_deque.PushBack(back\_deque.PopFront())

while not front\_deque.IsEmpty():

deque.PushFront(front\_deque.PopBack())

break

second\_item = deque.PopFront()

# Сравниваем элементы по x-координате

if first\_item[0] > second\_item[0]:

# Меняем элементы местами

deque.PushFront(first\_item)

back\_deque.PushBack(second\_item)

sorted = False

else:

deque.PushFront(second\_item)

back\_deque.PushBack(first\_item)

# Возвращаем элементы обратно в исходный дек

while not back\_deque.IsEmpty():

deque.PushFront(back\_deque.PopBack())

while not front\_deque.IsEmpty():

deque.PushFront(front\_deque.PopBack())

# Увеличиваем индекс

index += 1

if index + gap >= size:

break

# Сортируем пары координат методом расчески

comb\_sort(coordinate\_pairs)

def interpolate(x0, y0, x1, y1, x):

# Вычисляем наклон линии между двумя точками

try:

m = (y1 - y0) / (x1 - x0)

except ZeroDivisionError:

# print("Деление на ноль!")

m = y1 - y0

# Вычисляем интерполированное значение y в точке x

y = y0 + m \* (x - x0)

return y

def gauss\_legendre\_integration(x0, y0, x1, y1):

# Число точек для квадратуры Гаусса-Лежандра

n = 2

# Узлы для 2-точечной квадратуры Гаусса-Лежандра на интервале [-1, 1] (корни полинома Лежандра)

nodes = [-1.0 / math.sqrt(3), 1.0 / math.sqrt(3)]

# Веса для 2-точечной квадратуры Гаусса-Лежандра

weights = [1.0, 1.0]

# Половина длины интервала интегрирования

dx = (x1 - x0) / 2.0

# Середина интервала интегрирования

xm = (x1 + x0) / 2.0

# Инициализируем интеграл для текущего интервала

integral = 0.0

# Проходим по узлам квадратуры

for i in range(n):

# Преобразуем узел из интервала [-1, 1] в [x0, x1]

xi = xm + dx \* nodes[i]

# Интерполируем значение y в точке xi

yi = interpolate(x0, y0, x1, y1, xi)

# Добавляем взвешенное значение функции к интегралу

integral += weights[i] \* yi

# Умножаем на dx для получения значения интеграла на интервале

integral \*= dx

return integral

# Инициализируем общий интеграл равным нулю

total\_integral = 0.0

# Создаем временный дек для обхода coordinate\_pairs

temp\_deque = Deque()

# Проходим по парам координат для вычисления интеграла на каждом интервале

while not coordinate\_pairs.IsEmpty():

# Получаем начальную точку интервала

x0, y0 = coordinate\_pairs.PopFront()

# Копируем начальную точку в temp\_deque для последующего использования

temp\_deque.PushBack((x0, y0))

# Проверяем, есть ли следующая точка

if not coordinate\_pairs.IsEmpty():

# Получаем конечную точку интервала

x1, y1 = coordinate\_pairs.items[0] # Используем прямой доступ к первому элементу

# Вычисляем интеграл на текущем интервале методом Гаусса-Лежандра

integral = gauss\_legendre\_integration(x0, y0, x1, y1)

# Добавляем вычисленный интеграл к общему интегралу

total\_integral += integral

# Возвращаем элементы обратно в coordinate\_pairs

while not temp\_deque.IsEmpty():

coordinate\_pairs.PushFront(temp\_deque.PopBack())

# Выводим общий вычисленный интеграл

print("Интеграл равен:", total\_integral)

# Результаты тестирования

Тестирование производилось с помощью интерпретатора Python 3.13.0 и среды разработки [ms-python.python](https://marketplace.visualstudio.com/items?itemName=ms-python.python) версии 2024.3.10791010

Для заданных координат интеграл по методу Гаусса-Лежандра вычислен корректно.

Тест 1:

Входные данные:

**0 0**

**0 1**

**1 1**

**1 0**

Результат:



Тест 2:

Входные данные:

**0 1**

**0 2**

**0 3**

Результат:

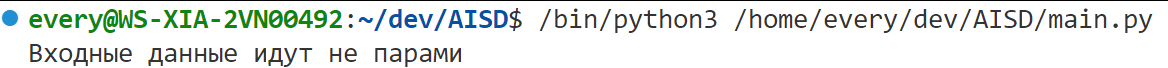


Тест 3:

Входные данные:

**London is the capital of Great Britain!**

Результат:



# Заключение

В ходе выполнения данной лабораторной работы была достигнута поставленная задача. А именно, был составлен и реализован алгоритм по сортировке пар координат расчёской, и подсчётом интеграла по заданным парам координат методом Гаусса-Лежандра.

В ходе решения данной задачи был реализован класс «Deque», реализующий структуру данных дек.

Также были реализованы функции «comb\_sort», «interpolate», «gauss\_legendre\_integration», соответственно реализующие алгоритм сортировки расчёской, интерполяцию значения y в точке x, и собственно интегрирование методом Гаусса-Лежандра.

Было выполнено тестирование программы на языке Python версии 3.13.0 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.