Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №2**

Дисциплина: «Разработка профессиональных приложений»

Двумерные структуры данных

Вариант 19

Выполнил:

студент группы ИВТАПбд-21

Романов Н.Ю

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту**

Вариант 19:

1. Выполнить обработку элементов квадратной матрицы A, имеющей N строк и N столбцов. Определить сумму элементов, расположенных параллельно главной диагонали (ближайшие к главной). Элементы главной диагонали имеют индексы от [0,0] до [N,N].
2. Ввод элементов списка должен быть доступен путем автоматической генерации. Необходимо использовать библиотеку numpy. Результаты выполнения должны сохраняться в файл (исходные данные и результат обработки).
3. Исходный код должен быть откомментирован
4. Необходимо реализовать правильную декомпозицию программы на методы.

**Описание реализации**

Для работы с массивом была использована библиотека numpy, в которой кроме базового варианта (многомерные массивы в базовом варианте) NumPy включает в себя набор пакетов для решения специализированных задач, например:

1. numpy.linalg — реализует операции линейной алгебры (простое умножение векторов и матриц есть в базовом варианте);
2. numpy.random — реализует функции для работы со случайными величинами;
3. numpy.fft — реализует прямое и обратное преобразование Фурье.

Массив создаётся с помощью метода .array(), в атрибутах которого можно указывать какого типа данные будут храниться в массиве, размерность массива, автозаполнение и прочее.

Случайное заполнения массива целыми числами в диапазоне от 0 до 10 выполняется благодаря методу .random.randint(0,10,size=(a,b)). В данном методе создаётся массив с a количеством строк и b количеством столбцов, которые заполняются случайными числами.

Листинг 1. Автоматическое заполнение массива

|  |
| --- |
| matrix = np.random.randint(10, size=(n, n)) |

Ручное заполнение массива осуществлено благодаря созданной отдельной функцией, в которую передаются значения количества строк и столбцов.

В данной функции создаётся временный двумерный массив, после чего считываются элементы построчно одной целой строкой, где все элементы разделяются пробелами. Это осуществляется с помощью цикла и методов .array() и .split(). То есть создаётся массив, в котором элементами являются разделённые пробелом числа введённой строки. После чего он записывается в i-тый элемент временного двумерного массива. После обхода всех строк функция возвращает полученный временный массив.

Листинг 2. Функция process\_matrix(matrix)

|  |
| --- |
| def process\_matrix(matrix):  diagonal\_indices = np.arange(matrix.shape[0])  parallel\_sum = sum(matrix[i, i + 1] + matrix[i + 1, i]  for i in diagonal\_indices[:-1])  return parallel\_sum |

Для реализации основного алгоритма программы были использованы методы .append() .split() и np.mean(). Метод .mean нужен для того чтобы вычислить среднее значения стоки или столбца, это делается с помощью указания axis(0 – столбец, 1-строка). После чего методом .append() в котором для того чтобы добавить новую строку нужно указать также через axis(0) а также т.к. np.mean() возвращает массив, то мы получаем нужную для нас строку. Но для того чтобы добавить новый столбец потребовалось массив полученный в результате выполнения np.mean() разделить методом .split(), который разделяет 1 массив на 3 разных объединяя их в список, т.к. .append() для столбцов требует отдельный элемент для добавления их в конец каждой строки.

Листинг 3. Функция save\_result(matrix, result):

|  |
| --- |
| def save\_result(matrix, result):  with open('result.txt', 'w')  as f: f.write(f"Matrix:\n{matrix}\n")  f.write(f"Parallel sum: {result}") |

Для вывода массива в отдельный файл была использована функция open(“output.txt”, “w+”) “w+” позволяет нам создать файл при его отсутствии, а также записать в него новые данные. Через метод .write() осуществляется сам ввод в файл. После чего методом .close() файл закрывается.

**Описание возникших затруднений**

В данной лабораторной работе возникли трудности с изучением различных методов из библиотеки numpy. Поскольку методов не малое количество, а также каждый метод является особенным.

**Описание альтернативных способов решения**

Алгоритм можно было реализовать немного иначе, с помощью применения списков вместо массивов, для которых бы потребовалось написать свой алгоритм нахождения среднего значения строки и столбца. Также массив можно было считывать с файла, для этого есть специально предусмотренный метод .load() из библиотеки numpy.

Тестирование

|  |
| --- |
|  |

Рис 1. Автоматический ввод массива

|  |  |
| --- | --- |
|  |  |

Рис 2. Ручной ввод массива

Вывод

В данной лабораторной работе были приобретены навыки владения библиотекой numpy, знания использования двумерных массивов на языке Python а также знания использования файлов. В результате выполнения была получена программа, которая выполняет обработку элементов квадратной матрицы A, имеющей N строк и N столбцов и определиет сумму элементов, расположенных параллельно главной диагонали.

**Код программы**

import numpy as np

def generate\_matrix(n) :

"""

Генерация квадратной матрицы размера n x n со случайными целочисленными значениями от 0 до 9

"""

return np.random.randint(10, size = (n, n))

def sum\_diagonal\_parallel(matrix) :

"""

Вычисление суммы элементов, расположенных параллельно главной диагонали(ближайшие к главной)

"""

diagonal\_sum = 0

for i in range(1, len(matrix)) :

# Суммируем элементы выше главной диагонали

diagonal\_sum += np.sum(np.diagonal(matrix, offset = i))

# Суммируем элементы ниже главной диагонали

diagonal\_sum += np.sum(np.diagonal(matrix, offset = -i))

# Суммируем элементы на главной диагонали

diagonal\_sum += np.sum(np.diagonal(matrix))

return diagonal\_sum

def save\_to\_file(filename, matrix, diagonal\_sum) :

"""

Сохранение исходных данных и результата обработки в файл

"""

with open(filename, 'w') as f :

f.write("Matrix:\n")

np.savetxt(f, matrix, fmt = "%d")

f.write("\n\nOriginal Matrix:\n")

np.savetxt(f, matrix, fmt = "%d")

f.write("\nSum of elements parallel to the main diagonal: ")

f.write(str(diagonal\_sum))

# Задаем размер квадратной матрицы

n = 5

# Генерируем матрицу

matrix = generate\_matrix(n)

# Вычисляем сумму элементов, расположенных параллельно главной диагонали

diagonal\_sum = sum\_diagonal\_parallel(matrix)

# Сохраняем исходные данные и результат обработки в файл

filename = "result.txt"

save\_to\_file(filename, matrix, diagonal\_sum)

# Выводим исходные данные и результат обработки в консоль

with open(filename, 'r') as f :

print(f.read())

# Выводим исходные данные и результат обработки в консоль

print("Matrix:")

print(matrix)

print("\nSum of elements parallel to the main diagonal:", diagonal\_sum)