Пояснительная записка к 3 домашнему заданию по предмету «Архитектура вычислительных систем» студента Дерели Серкана. Вариант 313.

Описание задания и выбранные параметры.

Описание задания.

По полученному варианту по формуле из описания задания был вычислен вариант задачи 1 = **5**, а также вариант задачи 2 — **23.**

1 часть задачи заключалась в написании обобщённой квадратной матрицы и 3 её конкретных реализаций: обычной квадратной матрицы с записью элементов в двумерный массив, диагональной матрицы с элементами в одномерном массиве, а также нижнетреугольной матрицы с элементами в одномерном массиве. У каждой конкретной реализации матрицы должен быть метод расчёта среднего арифметического элементов.

2 часть задачи заключалась в перемещении в начало контейнера тех матриц, чьё среднее арифметическое >= среднего арифметического средних арифметических значение элементов всех обобщённых квадратных матрицы в программе. Далее приведён кусок из условия задачи:

1 часть задания:

Обобщённый артефакт, используемый в задании.	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив).	Общие для всех альтернатив переменные.	Общие для всех альтернатив функции
5. Квадратные матрицы с действительными числами	1. Обычный двумерный массив 2. Диагональная (на основе одномерного массива) 3. Нижняя треугольная матрица (одномерный массив с формулой пересчета)	Размерность – целое число	

2 часть задания:

23. Переместить в начало контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

Выбранные случайные параметры, не очерченные в условии задачи.

В условии задачи чётко прописаны форматы тестовых данных: **файлы с тестовыми данными** могут содержать **не более 20 уникальных элементов** контейнера. Больше 20 элементов генерируется внутри программы случайным образом.

Однако для матриц не заданы граничные значения элементов и размера матрицы. Поэтому их выбор был сделан самостоятельно: **размер матрицы** задаётся **от 2 до 30** элементов в матрице включительно (тут имеется в виду параметр n матрицы. У диагональной матрицы n*n элементов будет n ненулевых элементов, у нижнетреугольной n*(n-1)/2 (по формуле суммы арифметической прогрессии) элементов, у обычной квадратной будет n^2 элементов).

Элементы же **матрицы** генерируются от **-100 до 100**. При чтении с файлов ограничения на читаемое число элементов нет.

Какие входные данные предполагаются программой верными.

Запуск программы предполагается из командной консоли на Линуксе. При этом в командной строке в функцию main программы передаются параметры работы программы — режим заполнения контейнера(с файла, случайно), путь к файлу с элементами контейнера или длина случайно генерируемого контейнера в зависимости от первого

параметра, а также путь файла с выходными данными. Длину контейнера при случайной генерации контейнера можно задать от 1 до 10 000 включительно(согласно условию). Чтобы входной файл с элементами контейнера правильно обрабатывался программой, элементы контейнера должны быть представлены в файле следующий образом:
Для диагональной матрицы:
<1> <Пробел><n>
<a1> <Пробел><a2><Пробел> <a3>...<an>
Для квадратной матрицы:
<2> <Пробел><n>
<a11> <Пробел><a12><Пробел> <a13>...<a1n>
<a21> <Пробел><a22><Пробел> <a23>...<a2n>
….....
<an1> <Пробел><na2><Пробел> <na3>...<ann>
Для диагональной матрицы:
<3> <Пробел><n>
<a11> <Пробел><a12><Пробел> <a13>...<a1(n*(n-1)/2)>

Среда разработки и инструменты разработки.

Для разработки программы использовалась среда Pycharm с интегрированным WSL(Ubuntu), язык python. Операционная система: Windows с установленным WSL(Ubuntu).

Структурная схема изучаемой архитектуры ВС с размещенной на ней разработанной программы.

Отображение содержимого классов на память.

Таблица классов	Таблица имён	Описание	
Container	__init__	func	def...
	getFromFile	func	def...
	generateRandom	func	def...
	print	func	def...
	write	func	def...
	rearrange	func	def...
	average	func	def...
	elements	list	[...]
Matrix	readElements	func	def...
	generateRandom	func	def...
	out	func	def...
	write	func	def...
	Average	func	def...
Diagonal(Matrix)	elements	list	[...]
	readElements	func	def...
	generateRandom	func	def...
	out	func	def...
	write	func	def...
	Average	func	def...

Square(Matrix)	elements	list	[...]
	readElements	func	def...
	generateRandom	func	def...
	out	func	def...
	write	func	def...
	Average	func	def...
Triangular(Matrix)	elements	list	[...]
	readElements	func	def...
	generateRandom	func	def...
	out	func	def...
	write	func	def...
	Average	func	def...

Отображение на память методов классов.

Память программы	Таблица имён	Память данных	
main.py	input_mode	str	<string>
	input_info	str	<string>
	output_info	str	<string>
	number_of_matrices	int	<number>
	input_file	file	fileName
	container	Container	container.py
	output_file	file	fileName
	errMessage1	func	def...

	errMessage2	func	def...
	errMessage3	func	def...
	errMessage4	func	def...
	container.py	module	container.py
Container.__init__	elements	list	[...]
Container.getFromFile	instream	file	fileName
	all_lines	list	[<string>, ...]
	container	Container	container.py
	i	int	<number>
	info_line	list	[<string>, <string>]
	matrix_type	str	<string>
	n	str	<string>
	matrix_type_num	int	<number>
	matrix	Diagonal or Square or Triangular	diagonal.py or square.py or triangular.py
	matrix.py	module	matrix.py
	diagonal.py	module	diagonal.py
	square.py	Module	square.py
	triangular.py	Module	triangular.py
Container.generateRandom	n	int	<number>
	container	Container	container.py
	i	int	<number>

	k	int	<number>
	Matrix	Diagonal or Square or Triangular	diagonal.py or square.py or triangular.py
Container.print	...	…	...
Container.write	…	…	...
Container.rearrange	…	…	...
Container.average	...	...	...
Matrix.readElements	...	...	...
Matrix.generateRandom	...	...	...
Matrix.out	...	...	...
Matrix.write	...	...	...
Matrix.average	...	...	...
Diagonal.readElements	...	...	...
Diagonal.generateRandom	...	...	...
Diagonal.out	...	...	...
Diagonal.write	...	...	...
Diagonal.average	...	...	...
Square.readElements	...	...	...
Square.generateRandom	...	...	...
Square.out	...	...	...
Square.write	...	...	...
Square.average	...	...	...
Triangular.readElements	...	...	...

Triangular.generateRandom	...	...	...
Triangular.out	...	...	...
Triangular.write	...	...	...
Triangular.average	...	...	...

Собираемые метрики программы.

Ниже представлены основные метрики программы:

Название файла	Размер файла .py	Размер файла .pyc
square	1 831 байт	2 098 байт
matrix	250 байт	886 байт
diagonal	1 337 байт	1 780 байт
triangular	1 533 байт	1 950 байт
container	3 998 байт	3 002 байт
rnd	241 байт	644 байт
main	3 108 байт	-

Файлов .py = 7. Файлов .pyc = 6

Общий размер файлов: 22658 байт

Тесты

Номер	Название и размер входного файла	Кол-во генерируемых элементов.	Название и размер выходного файла.	Сообщение об ошибке	Time prog total	Time prog user	Time prog sys
1	-	100	output100 361 798 байт	-	2164 ms	250 ms	156 ms
2	-	1000	output1000 3 446 528 байт	-	4604 ms	859 ms	438 ms
3	-	10000	output10000 35 453 620 байт	-	36363 ms	11438 ms	6594 ms
4	input0c 19 074 байт	-	output0c 12 750 байт	-	96 ms	31 ms	47 ms
5	input1c 3 961 байт	-	output1c 7 764 байт	-	78 ms	16 ms	63 ms
6	input2c 3734 байт	-	output2c 1 466 байт	-	73 ms	0 ms	63 ms
7	input3c 3346 байт	-	output3c 1 675 байт	-	71 ms	16 ms	63 ms
8	input4c 4655 байт	-	output4c 6 424 байт	-	82 ms	47 ms	31 ms
9	input5c	-	output5c	-	83 ms	16 ms	47 ms

	3110 байт		4 810 байт				
10	input6c 5156 байт	-	output6c 11 097 байт	-	91 ms	31 ms	63 ms
11	input7c 11186 байт	-	output7c 9 648 байт	-	88 ms	63 ms	16 ms
12	input8c 7629 байт	-	output8c 5 522 байт	-	83 ms	16 ms	31 ms
13	input9c 1480 байт	-	output9c 1 690 байт	-	80 ms	16 ms	47 ms
14	input10c 4570 байт	-	output10c 10 010 байт	-	84 ms	16 ms	31 ms
15	input11c 13885 байт	-	output11c 19 920 байт	-	104 ms	31 ms	47 ms
16	Input12w 0 байт	-	-	File is empty	66 ms	16 ms	47 ms
17	input13w 16925 байт	-	-	Wrong square matrix element. Couldn't fill the container.	70 ms	31 ms	31 ms

				Wront data in the file.			
18	input14w 7754 байт	-	-	Wrong square matrix element. Couldn't fill the container. Wront data in the file.	68 ms	31 ms	31 ms

Сравнение с прошлым дз.

Очевидно, что динамически типизированный язык проигрывает в скорости статически типизированным языкам. Скорость на тестовых данных в основном различалась в 1.5 — 2 раза в большую сторону у динамически типизированного питона. Размер исходных данных в программе на питоне получилась 22658 байт, в то время как во 2 дз, т. е. в Ооп подходе на c++ было 24092 байта, то есть сумма размеров исходных файлов сравнима. Безусловным преимуществом динамически типизированного языка является простота разработки, так как переменная может принимать значение любого типа, а значит их можно переиспользовать множество раз. Главные недостатком же является, конечно же, скорость. Она проседает заметно, но там, где скорость не критична, использование динамически типизированного языка оправдано. Собственно, так и происходит с питоном. Он хорош для анализа данных или написания прототипа приложения.

Вывод: динамически типизированный язык медленнее, но гибче и проще. Статически типизированный язык быстрее и надёжнее, но сложнее в использовании.