МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Реализовать программу, производящую сортировку матриц по возрастанию суммы чисел на главной диагонали с использованием алгоритма сортировки слиянием, и тестирование к ней.

Задание.

Сортировка слиянием. Python

На вход программе подаются квадратные матрицы чисел. Напишите программу, которая сортирует матрицы по возрастанию суммы чисел на главной диагонали с использованием алгоритма сортировки слиянием.

Формат входа.

Первая строка содержит натуральное число n - количество матриц. Далее на вход подаются n матриц, каждая из которых описана в формате: сначала отдельной строкой число mi - размерность i-й по счету матрицы. После m строк по m чисел в каждой строке - значения элементов матрицы.

Формат выхода.

Порядковые номера тех матриц, которые участвуют в слиянии на очередной итерации алгоритма. Вывод с новой строки для каждой итерации.

Массив, в котором содержатся порядковые номера матриц, отсортированных по возрастанию суммы элементов на диагонали. Порядковый номер матрицы - это её номер по счету, в котором она была подана на вход программе , нумерация начинается с нуля.

Пример

Вход:

3

2

1 2

1 31

3

1 1 1

1 11 1

1 1 -1

5

1 2 0 1 -1

1 2 0 1 -1

1 2 0 1 -1

1 2 0 1 -1

1 2 0 1 -1

Выход:

2 1

2 1 0

2 1 0

Для упрощения, можем свести задачу сортировки массива матриц к задаче сортировки массива чисел, где каждое число определяет сумму элементов диагонали матрицы.

При делении массива нечетной длины считаем, что первая часть после деления меньшая.

Выполнение работы.

Программа принимает на вход с клавиатуры число – количество матриц, которые следует отсортировать (*count_of_matrices*). Создаётся массив, который в дальнейшем будет подвергаться сортировке (*sorting_array*) и запускается цикл *for*, в каждой итерации которого в этот массив добавляется очередной элемент (список из двух элементов: порядковый номер матрицы и значение суммы чисел главной диагонали матрицы).

Для получения второго элемента списка вызывается функция *calculate_sums_of_main_diagonal*(). Она не принимает никаких аргументов и возвращает целое число - значение суммы чисел главной диагонали матрицы. Внутри этой функции матрица считывается с клавиатуры построчно (в цикле *for*). Чтобы задать условие цикла функция предварительно считывает через поток ввода размерность матрицы (*sums_of_main_diagonal*) и в том же цикле прибавляет

элемент главной диагонали со строки, считываемой в этой итерации.

Функция *print* печатает отформатированный результат работы рекурсивной функции *merge_sorting*. Она выполняет основную задачу – сортировку слиянием. Принимает в качестве аргумента массив, который нужно отсортировать. Если он состоит из одного элемента, функция возвращает его самого. В ином случае инициализируются три переменные: индекс центра массива (*center*), и индексы, хранящие порядковый номер элемента половины массива, на котором остановилась сортировка (*before_index* и *after_index*). Далее в два массива помещаются два отсортированных массива – результаты функции *merge_sorting* (т.е. запускается рекурсия) срезов разных половин исходного массива *(before_center* и *after_center*). Запускается цикл *while*, в котором сначала проверяются случаи, когда один из двух массивов для слияния закончился, и тогда в основной результирующий массив записывается остаток из второго, и наоборот. Иначе сравниваются текущие элементы массивов, и меньший записывается в результирующий. Индекс массива, элемент которого оказался наименьшим – сдвигается на один в большую сторону (алгоритм сортировки слиянием). После цикла в массив *intermediate_result* добавляется список порядковых номеров матриц, которые участвовали в слиянии на этой итерации алгоритма. Программа возвращает массив *done_array*, в которой перед этим записывается отсортированный на этой итерации рекурсии массив.

Затем идут функции тестирования программы.

Тестирование.

Написаны модульные тесты. Для этого реализован шаблон теста *pattern_test_for_sorting(test_array, answer),* который принимает на вход массив, который нужно отсортировать (элемент массива – список из двух элементов – порядковый номер и значение (в данной задаче – сумма чисел на главной диагонали матрицы)) и верный ответ теста в формате списка, где нулевой элемент - двумерный массив (его элемент – массив порядковых номеров элементов массива сортировки, которые участвуют в слиянии на

очередной итерации алгоритма), а первый является отсортированным массивом. Для отладки использована инструкция *assert*, параметром которой является текстовое сообщение, выводящее пользователю необходимую информацию об ошибке в случае её возникновения.

Далее реализована функция *test_for_sorting()*, вызываемая в коде программы один раз, производящая все заданные тестовые случаи. В случае успешного прохождения всех тестов программа также выводит соответствующее сообщение.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	[(0, 1), (1, 2), (2, 3)]	1 2 0 1 2 0 1 2	Программа работает верно
2.	[(0, 0), (1, 0)]	0 1 0 1	Программа работает верно
3.	[(0, 65), (1, -365), (2, 65), (3, -365)]	0 1 3 2 1 3 0 2 1 3 0 2	Программа работает верно
4.	3 2 1 2 1 31 3 1 1 1 1 11 1 1 1 -1 5 1 2 0 1 -1 1 2 0 1 -1 1 2 0 1 -1 1 2 0 1 -1 1 2 0 1 -1	2 1 2 1 0 2 1 0	Программа работает верно

Выводы.

Были изучены различные алгоритмы сортировок и принципы тестирования. Разработана программа, производящая сортировку матриц по возрастанию суммы чисел на главной диагонали с использованием алгоритма сортировки слиянием. Реализовано осуществлено модульное тестирование.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb2.py

```
intermediate_result = []
def merge_sorting(array):
    global intermediate_result
    done_array = []
    if len(array) == 1:
        return array
    center = len(array) // 2
    before_index = 0
    after_index = 0
    before_center = merge_sorting(array[:center])
    after_center = merge_sorting(array[center:])
    while (before_index < len(before_center) or
    after_index < len(after_center)):
        if before_index == len(before_center):

    done_array.extend(after_center[after_index:])
            after_index = len(after_center)
        elif after_index == len(after_center):

    done_array.extend(before_center[before_index:])
            before_index = len(before_center)
        elif before_center[before_index][1] <=
    after_center[after_index][1]:

    done_array.append(before_center[before_index])
            before_index += 1
        elif before_center[before_index][1] >
    after_center[after_index][1]:

    done_array.append(after_center[after_index])
            after_index += 1
    intermediate_result.append([j[0] for j in
    done_array])
```

```
    print(" ".join(map(lambda a: str(a[0]),
    done_array)))
    return done_array


def calculate_sums_of_main_diagonal():
    sums_of_main_diagonal = 0
    dimension_of_matrix = int(input())
    for diagonal_element in
    range(dimension_of_matrix):
        sums_of_main_diagonal +=
    int(input().split()[diagonal_element])
    return sums_of_main_diagonal


count_of_matrices = int(input())
sorting_array = []
for matrix_number in range(count_of_matrices):
    sorting_array.append((matrix_number,
    calculate_sums_of_main_diagonal()))
print(" ".join(map(lambda a: str(a[0]),
    merge_sorting(sorting_array))))

def pattern_test_for_sorting(test_array, answer):
    global intermediate_result
    intermediate_result.clear()
    assert [j[0] for j in merge_sorting(test_array)]
    == answer[1] and intermediate_result == answer[0],
    "\nTest: {}\nGot: {}\nExpected: {}"
    .format(test_array, [j[0] for j in
    merge_sorting(test_array)],answer[1])


def test_for_sorting():
    test_array = [(0, 1), (1, 2), (2, 3)]
    answer = ([[1, 2], [0, 1, 2]], [0, 1, 2])
    pattern_test_for_sorting(test_array, answer)
```

```
    test_array = [(0, 0), (1, 0)]
    answer = ([[0, 1]],[0, 1])
    pattern_test_for_sorting(test_array, answer)

    test_array = [(0, 65), (1, -365), (2, 65), (3, -
    365)]
    answer = ([[1, 0], [3, 2], [1, 3, 0, 2]], [1, 3,
    0, 2])
    pattern_test_for_sorting(test_array, answer)

test_for_sorting()

print("All tests were successfully passed")
```