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. 1304 _____ Чернякова В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2022

Цель работы.

Освоить работу алгоритма, осуществляющего сортировку способом слияния.

Задание.

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 m_i - 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

Объяснение:

n = 3

$m_0 = 0$

Первая матрица (порядковый номер 0):

1 2

1 31

$m_1 = 3$

Вторая матрица (порядковый номер 1):

1 1 1

1 11 1

1 1 -1

$m_2 = 5$

Третья матрица (порядковый номер 2):

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

Сумма элементов диагонали матрицы с порядковым номером 0 = 32
Сумма элементов диагонали матрицы с порядковым номером 1 = 11
Сумма элементов диагонали матрицы с порядковым номером 2 = 3
Для упрощения, можем свести задачу сортировки массива матриц к задаче сортировки массива чисел, где каждое число определяет сумму элементов диагонали матрицы. В итоге мы имеем массив элементов с порядковыми номерами [0, 1, 2] и суммой элементов на главной диагонали 32, 11, 3 для нулевого, первого и второго по порядку элементов соответственно. На первой итерации сортировки исходный массив делится на два полмассива:

[0]

и

[1, 2]

Далее происходит деление второго массива на два массива по одному элементу:

[1]

[2]

После происходит слияние. Массивы [1] и [2] сливаются в один:

[2, 1]

Порядок элементов такой, поскольку сумма элементов диагонали матрицы с порядковым номером 2 меньше, чем сумма элементов диагонали матрицы с порядковым номером 1.

В этот момент ваша программа должна сделать первый вывод. Вывод содержит только порядковые номера матриц, разделенные пробелом. Далее массив [2, 1] сливается с массивом [0]:

[2, 1, 0]

И это является вторым выводом.
Массив отсортирован, теперь нужно вывести окончательный результат сортировки:

2 1 0.

Поэтому, правильный вывод задачи выглядит так:

2 1

2 1 0

2 1 0

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

Примечание: *вы можете использовать библиотеку numpy, но это не является обязательным.*

Выполнение работы.

На вход программе с помощью функции *input()* подается строка, содержащая натуральное число *n* – количество матриц. С помощью функции *int()* введённая строка преобразуется в число. Далее с помощью функции *list()* создается список *summaDM*, в котором будут храниться значения суммы чисел на главной диагонали матриц. Также с помощью функции *list()* для корректного вывода ответа на задание лабораторной работы создается список *all_result*, в котором будет храниться результат работы основной функции *merge()*.

Циклом *for* от *0* до *n* не включительно переменной *ind* осуществляется перебор. В теле цикла обнуляется переменная *summaD*, в которой будет храниться сумма элементов на главной диагонали введённой матрицы. С помощью функции *input()* считывается строка *mi* - размерность *ind*-й по счету матрицы, тип которой преобразуются к целочисленному *int()*. Обнуляется переменная *indForSum* – отвечает за индекс элемента на главной диагонали матрицы.

Считывание данных каждой матрицы происходит циклом *for* от *0* до значения *mi* не включительно переменной *j*. Благодаря *list()* формируется список *line*, в котором хранится строка обрабатываемой на данной итерации матрицы. Значение *summaD* увеличивается на число, хранящемся в списке line по индексу *indForSum*. *indForSum* увеличивается на единицу для перехода к следующему диагональному элементу. В список *summaDM* с помощью

функции *append()* добавляется список из двух элементов – *summaDM*(сумма диагональных элементов матрицы), *ind*(индекс обрабатываемой матрицы).

Переменной *answer* присваивается значение работы функции merge. Циклом *for* на экран выводится обработанный список *answer* согласно заданным значениям параметра выхода данной лабораторной работы.

Функции.

Функция *def merge(arr, arr_result)* принимает на вход в качестве аргументов список, в котором хранятся индексы введенных матриц и соответствующие им суммы диагональных элементов и список, в котором будет храниться результат работы функции. В теле функции обусловлена обработка базового случая – *if len(arr) == 1: return*. Индексу деления списка пополам *middle* присваивается значение *len(arr) // 2*. Создается 2 отдельных списка: левая *left* и правая *right* часть с помощью срезов в среде программирования *Python arr[:middle], arr[middle:]*. По определению функция вызывается для левой и правой части: *merge(left, arr_result)* и *merge(right, arr_result)*.Для осуществления сравнения создаются переменные-индексы, отвечающие за перемещения по спискам левой и правой части соответственно, а также индекс, ходящий по списку с результатами *index_left = index_right = index = 0*. Результирующий список соответствующей длины *result = [0] * (len(left) + len(right))*. С помощью функции *list()* создается список *merge_list*, в котором будут храниться порядковые номера тех матриц, которые участвуют в слиянии на очередной итерации алгоритма.

С помощью цикла *while* осуществляется проход по левой и правой части до тех пор, пока одна из них не закончится: *index_left < len(left) and index_right < len(right)*. Осуществляется сравнение элементов двух частей *if left[index_left][0] <= right[index_right][0]* иначе работает блок *else* и наименьшее значение записывается по индексу в результирующий список. При выполнении первого условия выполняется цепочка действий: *result[index] = left[index_left], index_left += 1*. Иначе *result[index] = right[index_right]*,

index_right += 1. В конце значение результирующего *index* увеличивается на единицу.

В случае если левая часть не закончилась запускается цикл *while* до тех пор, пока *index_left < len(left)*. В результирующий список добавляется соответствующий элемент левой части *result[index] = left[index_left]*. Индексы левой части и результирующего списка увеличиваются на единицу.

В случае если правая часть не закончилась запускается цикл while до тех пор, пока *index_right < len(right)*. В результирующий список добавляется соответствующий элемент правой части *result[index] = right[index_right]*. Индексы правой части и результирующего списка увеличиваются на единицу.

Для записи в список *arr* отсортированного массива с помощью цикла *for* осуществляется проход, и соответствующим значениям по индексу *i* в этом массиве записываются значения по тому же индексу в списке *result*. Также в список *merge_list* с помощью функции *append()* добавляется значение – *arr[i][1],* то есть индекс матрицы, участвующей в слиянии.

В список *arr_result*, хранящий результат работы функции на каждой итерации, добавляется получившийся на данном этапе список *merge_list*. Функция возвращает получившийся результирующий список return *arr_result*.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	4 2 3 5 12 6 1 4 3 12 45 6	1 0 3 2 3 1 0 2 3 1 0 2	Проверка работы алгоритма для четного количества матриц.

	12 34 6 1 5 7 2 1 1 0 1		
2.	3 1 23 3 12 -45 23 12 2 4 0 0 0 3 1 0 0 12 3 5 -23 5 -6	2 1 2 1 0 2 1 0	Проверка работы алгоритма для нечетного количества матриц.
3.	5 2 1 6 23 8 1 9 3 -5 4 6 0 10 4 12 4 4 3 18 2 54 2 -9 1 1 1 0 2 9 0 0 9	0 1 3 4 2 3 4 0 1 2 3 4 0 1 2 3 4	Проверка работы алгоритма для матриц, у которых сумма чисел на диагонали одинаковая.

4.	3 2 -23 4 12 -3 2 100 12 0 -231 3 1 0 0 0 1 0 0 0 1	1 2 1 0 2 1 0 2	Проверка работы алгоритма при наличии отрицательных сумм на диагонали матриц.

Выводы.

Был изучен алгоритм сортировки слиянием. На основе данного алгоритма была создана программа. Написано тестирование для программного кода, проверяющее его корректность.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
def merge(arr, arr_result):
    if len(arr) == 1:
        return
    middle = len(arr) // 2
    left, right = arr[:middle], arr[middle:]
    merge(left, arr_result)
    merge(right, arr_result)
    index_left = index_right = index = 0
    result = [0] * (len(left) + len(right))
    merge_list = list()
    while index_left < len(left) and index_right < len(right):
        if left[index_left][0] <= right[index_right][0]:
            result[index] = left[index_left]
            index_left += 1
        else:
            result[index] = right[index_right]
            index_right += 1
        index += 1
    while index_left < len(left):
        result[index] = left[index_left]
        index_left += 1
        index += 1
    while index_right < len(right):
        result[index] = right[index_right]
        index_right += 1
        index += 1
    for i in range(len(arr)):
        arr[i] = result[i]
        merge_list.append(arr[i][1])
    arr_result.append(merge_list)
    return arr_result

if __name__ == '__main__':
    n = int(input())
```

```
    summaDM = list()
    arr_result = list()
    for ind in range(n):
        summaD = 0
        mi = int(input())
        indForSum = 0
        for j in range(mi):
            line = list(map(int, input().split()))
            summaD += line[indForSum]
            indForSum += 1
        summaDM.append((summaD, ind))
    answer = merge(summaDM, arr_result)
    for ind in range(len(answer)):
        for j in answer[ind]:
            print(j, end=' ')
        print()
        if ind == len(answer) - 1:
            for j in answer[ind]:
                print(j, end=' ')
```

Название файла: test.py

```
from main import merge
import pytest

@pytest.mark.parametrize("arr, arr_result, expected_result",
[([(35, 0), (-282, 1), (83, 2)], [], [[1, 2], [1, 0, 2]]),
([(9, 0), (4, 1), (53, 2), (2, 3)], [], [[1, 0], [3, 2], [3, 1, 0,
2]]),
([(23, 0), (14, 1), (-2, 2)], [], [[2, 1], [2, 1, 0]]),
([(9, 0), (9, 1), (9, 2), (9, 3), (18, 4)], [], [[0, 1], [3, 4],
[2, 3, 4], [0, 1, 2, 3, 4]]),
([(-26, 0), (-131, 1), (3, 2)], [], [[1, 2], [1, 0, 2]])
])

def test(arr, arr_result, expected_result):
     assert merge(arr, arr_result) == expected_result
```