**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 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

**Объяснение:**

n = 3

m0 = 0

Первая матрица (порядковый номер 0):

1 2

1 31

m1 = 3

Вторая матрица (порядковый номер 1):

1 1 1

1 11 1

1 1 -1

m2 = 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 подается целое число *n*, равное количеству вершин дерева, и список родителей, введенный через пробел.

Функции.

Функция *def tree\_creator(data)* принимает на вход в качестве аргумента список, в котором хранятся *n* целых чисел, являющихся родителями вершин дерева. В теле функции создается список *tree* с помощью встроенной функции *list()*. Так как в обрабатываемом списке значения – строки, то с помощью *int()* онипреобразуются в целые числа для дальнейшей работы. Функция с помощью *return* возвращает созданный список.

Функция *def height(tree, n)* принимает на вход в качестве аргументов список с родителями вершин дерева и количество вершин. С помощью условного оператора *if* проверяется количество вершин. Если она одна, то функция с помощью *return* возвращается значение *n*, равное количеству вершин. Иначе работает блок *else*. Создается список *all\_height* длинной *n* и начальными значениями *0*, в нем будут хранится все возможные высоты дерева. С помощью цикла *for* перебираются все вершины от *0* до *n* не включительно. Переменной *current* присваивается значение *i*, вершины, на данном шаге цикла. Далее с помощью цикла *while* перебираются все вершины-родители, связанные с изначальной, пока не будет достигнута *-1*, то есть корень. Для вычисления высоты необходимо перейти к родителю вершины, поэтому переменной *parent* присваивается значение из списка *tree*, которое находится под индексом текущего значения *current*. В *current* теперь находится вершина-родитель. Значение в списке *all\_height* по индексу *i* увеличивается на *1* – значение высоты дерева относительно вершины, обрабатываемой на данной итерации цикла. Далее с помощью условного оператор *if* проверяется, есть ли уже значение высоты в списке *all\_height* по индексу *parent* – родителя, если да, то значение в списке *all\_height* по индексу *i* увеличивается на значение, проверяемое условием – полная высота. Функция с помощью *return* возвращает максимальное значение в списке высот - *max(all\_height).*

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  -1 | 1 | Проверка крайнего случая – на вход подан только корень дерева. |
|  | 3  -1 0 0 | 2 | Проверка работы алгоритма для дерева из двух веток. |
|  | 5  3 4 3 4 -1 | 3 | Работа программы корректна. |
|  | 9  4 0 8 -1 3 7 2 1 5 | 9 | Проверка корректного работы алгоритма, когда у дерева только одна ветка. |
|  | 16  15 2 4 2 8 6 4 6 -1 10 12 10 8 14 12 14 | 5 | Корректная работа программы для произвольного большого дерева. |

## Выводы.

Был изучен алгоритм нахождения высоты дерева. На основе данного алгоритма была создана программа. Освоена работа с pytest и написано тестирование для программного кода, проверяющее его корректность.

# Приложение А Исходный код программы

Название файла: main.py

def tree\_creator(data):

tree = list(int(parents) for parents in data)

return tree

def height(tree, n):

if n == 1:

return n

else:

all\_height = n\*[0]

for i in range(n):

current = i

while current != -1:

parent = tree[current]

current = parent

all\_height[i] += 1

if all\_height[parent]:

all\_height[i] += all\_height[parent]

break

return max(all\_height)

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input())

parents = list(map(int, input().split()))

print(height(tree\_creator(parents), n))

Название файла: test.py

from main import height

import pytest

@pytest.mark.parametrize("tree, n, expected\_result",

[([-1], 1, 1),

([-1, 0, 0], 3, 2),

([3, 4, 3, 4, -1], 5, 3),

([4, 0, 8, -1, 3, 7, 2, 1, 5], 9, 9),

([15, 2, 4, 2, 8, 6, 4, 6, -1, 10, 12, 10, 8, 14, 12, 14], 16, 5)])

def test(tree, n, expected\_result):

assert height(tree, n) == expected\_result