**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Сортировки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0381 |  | Дзаппала Д. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2021

## Цель работы.

Ознакомиться с алгоритмом сортировки слиянием, и реализовать алгоритм сортировки матриц по сумме чисел на их диагонали.

## 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

## Выполнение работы.

В самом начале идет считывание кол-ва матриц, которое нам подадут на вход. После чего, был создан std::vector<std::pair<int, int>> (в программе используется псевдоним для std::pair<int, int> = matrixIndexSum), в который и заносилась сумма, которая считалась при считываний матриц. Функция MergeSort принимает на вход 3 аргумента: сам вектор (arr) и два начальных диапазона, left и right (0 и arr.size, соответственно). Алгоритм рекурсивный, поэтому вначале функции идет проверка диапазона, сколько в нашем «подмассиве» элементов (1). В случае, если одни эл-т, то возвращаемся, иначе идет рассчет середины «подмассива», и рекурсивно вызывается эта же функция. После того, как функции отработали, и мы получили отсортированные подмассивы, вызывается функция Merge, принимающая 4 аргумента: массив, left, mid и right, для диапазонов. Функция Merge создает временный «вектор», в который запихиваются эл-мы в соответсвии с тем, у кого больше сумма (диагонали матрицы). То есть, первый цикл работает, пока для одного из подмассивов мы в его диапазоне (для левого — [left, mid), для правого — [mid, right)). После чего еще раз запускаем циклы для обоих подмассивов. Они нужны на тот случай, если изначальный массив был нечетной длины, и какой-то подмассив содержит нечетное кол-во элементов. После чего, временный вектор переписывается вначальный массив. Когда вся рекурсия отработает, мы получаем отсортированный массив.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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 |  |
|  | 3  2  -62 -8  -1 97  3  -98 -84 28  32 -85 -33  96 -68 -99  2  15 81  67 68 | 1 2  1 0 2  1 0 2 |  |

## Выводы.

Был изучен алгоритм сортировки слиянием.

# Приложение А Исходный код программы

Название файла: src.cpp

#include<iostream>

#include<vector>

using matrixIndexSum = std::pair<int, int>;

#include<cassert>

void Merge(std::vector<matrixIndexSum>& arr, int left, int mid, int right){

int indL = 0; int indR = 0;

std::vector<matrixIndexSum> res(right - left);

while( left + indL < mid && mid + indR < right){

if (arr[left+indL].second <= arr[mid+indR].second){

res[indL+indR] = arr[left+indL];

indL++;

}

else{

res[indL+indR] = arr[mid+indR];

indR++;

}

}

while(left+indL < mid){

res[indL+indR] = arr[left+indL];

indL++;

}

while (mid + indR < right){

res[indL+indR] = arr[mid+indR];

indR++;

}

for (int i = 0; i < indL+indR; i++){

arr[left+i] = res[i];

}

}

void MergeSort(std::vector<matrixIndexSum>& arr, int left, int right){

if (left + 1 >= right)

return;

int mid = (left + right) / 2;

MergeSort(arr, left, mid);

MergeSort(arr, mid, right);

Merge(arr, left, mid, right);

// for (int i = left; i < right; i++){

// std::cout << arr[i].first << " ";

// }

// std::cout << std::endl;

}

void TESTS(){

using test = std::vector<matrixIndexSum>;

using correct = std::vector<matrixIndexSum>;

int testC = 1;

std::cout << "Test#" << testC++ << ": ";

test t1 = {{0, 12}, {1, 2}, {2, 5}, {3, 1}, {4, 7}};

correct c1 = {{3, 1}, {1, 2}, {2, 5}, {4, 7}, {0, 12}};

MergeSort(t1, 0, t1.size());

assert(t1 == c1 && "incorrect!");

std::cout << "correct" << std::endl;

std::cout << "Test#" << testC++ << ": ";

test t2 = {{0, 1}, {1, 2}, {2, 10}, {3, 2}};

correct c2 = {{0, 1}, {1, 2}, {3, 2}, {2, 10}};

MergeSort(t2, 0, t2.size());

assert(t2 == c2 && "incorrect!");

std::cout << "correct" << std::endl;

std::cout << "Test#" << testC++ << ": ";

test t3 = {{0, 1}};

correct c3 = {{0, 1}};

MergeSort(t3, 0, t3.size());

assert(t3 == c3 && "incorrect!");

std::cout << "correct" << std::endl;

std::cout << "Test#" << testC++ << ": ";

test t4 = {{0, -10}, {1, 10}, {2, -200}, {3, 15}, {4, 0}, {5, 17}};

correct c4 = {{2, -200}, {0, -10}, {4, 0}, {1, 10}, {3, 15}, {5, 17}};

MergeSort(t4, 0, t4.size());

assert(t4 == c4 && "incorrect!");

std::cout << "correct" << std::endl;

}

int main(){

// int matrixCount;

// std::cin >> matrixCount;

// std::vector<matrixIndexSum> matrixArray(matrixCount);

// for (int matrC = 0; matrC < matrixCount; matrC++){

// int matrixDimension;

// std::cin >> matrixDimension;

// int diagonalSum = 0, tmpMatrEl;

// for (int matrixRow = 0; matrixRow < matrixDimension; matrixRow++){

// for (int matrixColoumn = 0; matrixColoumn < matrixDimension; matrixColoumn++){

// std::cin >> tmpMatrEl;

// if (matrixRow == matrixColoumn){

// diagonalSum += tmpMatrEl;

// }

// }

// }

// matrixArray[matrC] = std::make\_pair(matrC, diagonalSum);

// }

// MergeSort(matrixArray, 0, matrixArray.size());

// for (int i = 0; i < matrixArray.size(); i++){

// std::cout << matrixArray[i].first << " ";

// }

// std::cout << std::endl;

TESTS();

return 0;

}