Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

З дисципліни «ПРО»

Виконав: Перевірив:

Студент групи ІО-21 доц. Корочкін О.В.

Коноз А.О.

Дата здачі\_\_\_\_\_\_\_\_\_\_\_\_\_

Захищено з балом\_\_\_\_\_

Київ 2014

**Завдання**

1.23 2.13 3.17

1. F1: E:=A+B+C+D(MA\*MZ).

2. F2: MZ:=Min(MA)\*MB+Max(MT)\*(MX\*MZ).

3. F3: d:=Min(A\*Trans(MB\*MM)+B\*Sort(MC).

**Лістинг коду**

**import** java.util.Scanner;

/\*\*

\* **@author** User

\* Головний потік програми

\*/

**public** **class** Lab1 {

/\*\*

\*

\*/

**public** **static** **void** main(String[] args) **throws** InterruptedException{

System.*out*.println("Main thread started");

System.*out*.println("Enter n: ");

Scanner scanner = **new** Scanner(System.*in*);

**int** n = scanner.nextInt();

TF1 tf1 = **new** TF1(n, 100, 2, "TF1");

TF2 tf2 = **new** TF2(n, 1000, 5, "TF2");

TF3 tf3 = **new** TF3(n, 1500, 8, "TF3");

tf1.start();

tf2.start();

tf3.start();

tf1.join();

tf2.join();

tf3.join();

System.*out*.println("Main thread finished");

}

}

/\*\*

\* **@author** User

\* Потік для обчислення функції:

\* F1: E:=A+B+C+D(MA\*MZ)

\*/

**public** **class** TF1 **extends** Thread {

**private** **double** vectA[];

**private** **double** vectB[];

**private** **double** vectC[];

**private** **double** vectD[];

**private** **double** matrA[][];

**private** **double** matrZ[][];

**private** **int** n;

**private** **int** sleep;

/\*\*

\* **@param** n кількість елементів

\* **@param** sleep час затримки

\* **@param** priority пріоритет потоку

\* **@param** name назва потоку

\*/

**public** TF1(**int** n, **int** sleep, **int** priority, String name){

**this**.setName(name);

**this**.setPriority(priority);

**this**.n = n;

**this**.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

**public** **void** run(){

**double** vectRes[];

System.*out*.println(**this**.getName() +" started");

vectA = Tools.*readVect*(n, "Enter vector A for F1");

vectB = Tools.*readVect*(n, "Enter vector B for F1");

vectC = Tools.*readVect*(n, "Enter vector C for F1");

vectD = Tools.*readVect*(n, "Enter vector D for F1");

matrA = Tools.*readMatr*(n, "Enter matrix A for F1");

matrZ = Tools.*readMatr*(n, "Enter matrix Z for F1");

**try** {

Thread.*sleep*(sleep);

} **catch** (Exception e) {

}

vectRes = Tools.*addVect*(Tools.*addVect*(vectA,vectB),Tools.*addVect*(vectC,Tools.*multVectMatr*(vectD,Tools.*multMatr*(matrA,matrZ))));

Tools.*showVect*(vectRes, "Result of F1");

System.*out*.println(**this**.getName() + " finished");

}

}

/\*\*

\* **@author** User

\* Потік для обчислення функції:

\* F2: MZ:=Min(MA)\*MB+Max(MT)\*(MX\*MZ)

\*/

**public** **class** TF2 **extends** Thread{

**private** **double** matrA[][];

**private** **double** matrB[][];

**private** **double** matrM[][];

**private** **double** matrX[][];

**private** **double** matrZ[][];

**private** **int** n;

**private** **int** sleep;

/\*\*

\* **@param** n кількість елементів

\* **@param** sleep час затримки

\* **@param** priority пріоритет потоку

\* **@param** name назва потоку

\*/

**public** TF2(**int** n, **int** sleep, **int** priority, String name){

**this**.setName(name);

**this**.setPriority(priority);

**this**.n = n;

**this**.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

**public** **void** run(){

**double** matrRes[][];

System.*out*.println(**this**.getName() +" started");

matrA = Tools.*readMatr*(n, "Enter matrix A for F2");

matrB = Tools.*readMatr*(n, "Enter matrix B for F2");

matrM = Tools.*readMatr*(n, "Enter matrix M for F2");

matrX = Tools.*readMatr*(n, "Enter matrix X for F2");

matrZ = Tools.*readMatr*(n, "Enter matrix Z for F2");

**try** {

Thread.*sleep*(sleep);

} **catch** (Exception e) {

// **TODO**: handle exception

}

matrRes = Tools.*addMatr*(Tools.*multNumbMatr*(Tools.*findMinMatr*(matrA),matrB),Tools.*multNumbMatr*(Tools.*findMaxMatr*(matrM),Tools.*multMatr*(matrX,matrZ)));

Tools.*showMatr*(matrRes, "Result of F2");

System.*out*.println(**this**.getName() + " finished");

}

}

/\*\*

\* **@author** User

\* Потік для обчислення функції:

\* F3: d:=Min(A\*Trans(MB\*MM)+B\*Sort(MC))

\*/

**public** **class** TF3 **extends** Thread{

**private** **double** vectA[];

**private** **double** vectB[];

**private** **double** matrB[][];

**private** **double** matrM[][];

**private** **double** matrC[][];

**private** **int** n;

**private** **int** sleep;

/\*\*

\* **@param** n кількість елементів

\* **@param** sleep час затримки

\* **@param** priority пріоритет потоку

\* **@param** name назва потоку

\*/

**public** TF3(**int** n, **int** sleep, **int** priority, String name){

**this**.setName(name);

**this**.setPriority(priority);

**this**.n = n;

**this**.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

**public** **void** run(){

**double** result;

System.*out*.println(**this**.getName() + " started");

vectA = Tools.*readVect*(n, "Enter vector A for F3");

vectB = Tools.*readVect*(n, "Enter vector B for F3");

matrB = Tools.*readMatr*(n, "Enter matrix B for F3");

matrM = Tools.*readMatr*(n, "Enter matrix M for F3");

matrC = Tools.*readMatr*(n, "Enter matrix C for F3");

**try** {

Thread.*sleep*(sleep);

} **catch** (Exception e) {

// **TODO**: handle exception

}

result = Tools.*findMinVect*(Tools.*addVect*(Tools.*multVectMatr*(vectA,Tools.*transpMatr*(Tools.*multMatr*(matrB,matrM))),Tools.*multVectMatr*(vectB,Tools.*sortMatr*(matrC))));

System.*out*.println("Result of F3: " + result);

System.*out*.println(**this**.getName() + " finished");

}

}

**import** java.io.InputStreamReader;

**import** java.util.Scanner;

/\*\*

\* **@author** User

\* Клас з статичними універсальними методами

\*/

**public** **class** Tools {

/\*\*

\* Додає два вектори

\* **@param** vectA вектор

\* **@param** vectB вектор

\* **@return** вектор суми

\*/

**public** **static** **double**[] addVect(**double** vectA[], **double** vectB[]){

**for** (**int** i = 0; i < vectB.length; i++) {

vectA[i] += vectB[i];

}

**return** vectA;

}

/\*\*

\* Перемножає дві матриці

\* **@param** matrA матриця

\* **@param** matrB матриця

\* **@return** повертає добуток двох матриць

\*/

**public** **static** **double**[][] multMatr(**double** matrA[][], **double** matrB[][]){

**double** temp = 0;

**double** resultMatr[][] = **new** **double**[matrA.length][matrA[1].length];

**for** (**int** i = 0; i < matrA.length; i++) {

**for** (**int** j = 0; j < matrA[i].length; j++) {

temp = 0;

**for** (**int** k = 0; k < matrB.length; k++) {

temp += matrA[i][k] \* matrB[k][j];

}

resultMatr[i][j] = temp;

}

}

**return** resultMatr;

}

/\*\*

\* Транспонує матрицю

\* **@param** matrA матриця для транспонування

\* **@return** транспоновану матрицю

\*/

**public** **static** **double**[][] transpMatr(**double** matrA[][]){

**double** resultMatr[][] = **new** **double** [matrA[0].length][matrA.length];

**for** (**int** i = 0; i < resultMatr.length; i++) {

**for** (**int** j = 0; j < resultMatr[0].length; j++) {

resultMatr[i][j] = matrA[j][i];

}

}

**return** resultMatr;

}

/\*\*

\* Знаходить максимальний елемент в матриці

\* **@param** matrA матриця для пошуку

\* **@return** максимальний елемент матриці

\*/

**public** **static** **double** findMaxMatr(**double** matrA[][]){

**double** max = matrA[0][0];

**for** (**int** i = 0; i < matrA.length; i++) {

**for** (**int** j = 0; j < matrA.length; j++) {

**if**(max < matrA[i][j]){

max = matrA[i][j];

}

}

}

**return** max;

}

/\*\*

\* Знаходить мінімальний елемент матриці

\* **@param** matrA матриця для пошуку

\* **@return** мінімальний елемент

\*/

**public** **static** **double** findMinMatr(**double** matrA[][]){

**double** min = matrA[0][0];

**for** (**int** i = 0; i < matrA.length; i++) {

**for** (**int** j = 0; j < matrA.length; j++) {

**if**(min > matrA[i][j]){

min = matrA[i][j];

}

}

}

**return** min;

}

/\*\*

\* Знаходить мінімальний елемент в векторі

\* **@param** vectA вектор для пошуку

\* **@return** мінімальний елемент

\*/

**public** **static** **double** findMinVect(**double** vectA[]){

**double** min = vectA[0];

**for** (**int** i = 0; i < vectA.length; i++) {

**if**(min > vectA[i]){

min = vectA[i];

}

}

**return** min;

}

/\*\*

\* Додає дві матриці

\* **@param** matrA матриця

\* **@param** matrB матриця

\* **@return** суму матриць

\*/

**public** **static** **double** [][] addMatr(**double** matrA[][], **double** matrB[][]){

**for** (**int** i = 0; i < matrB.length; i++) {

**for** (**int** j = 0; j < matrB[0].length; j++) {

matrA[i][j] += matrB[i][j];

}

}

**return** matrA;

}

/\*\*

\* Перемножає вектор на матрицю

\* **@param** vectA вектор

\* **@param** matrB матрицю

\* **@return** добуток вектора і матриці

\*/

**public** **static** **double**[] multVectMatr(**double** vectA[], **double** matrB[][]){

**double** temp = 0;

**double** resultVect[] = **new** **double**[vectA.length];

**for** (**int** i = 0; i < matrB.length; i++) {

temp = 0;

**for** (**int** j = 0; j < resultVect.length; j++) {

temp += vectA[j]\*matrB[j][i];

}

resultVect[i] = temp;

}

**return** resultVect;

}

/\*\*

\* Сортує вектор за зростанням

\* **@param** vectA вектор для сортування

\* **@return** відсортований вектор

\*/

**public** **static** **double**[] sortVect(**double** vectA[]){

**double** temp = 0;

**for** (**int** i = 0; i < vectA.length; i++) {

**for** (**int** j = 0; j < (i-1); j++) {

**if**(vectA[j] > vectA[j+1]){

temp = vectA[j];

vectA[j] = vectA[j+1];

vectA[j+1] = temp;

}

}

}

**return** vectA;

}

/\*\*

\* Сортує матрицю за спаданням

\* **@param** matrA матриця для сортування

\* **@return** відсортовану матрицю

\*/

**public** **static** **double**[][] sortMatr(**double** matrA[][]){

**double** temp = 0;

**for** (**int** k = 0; k < matrA.length; k++) {

**for** (**int** i = 0; i < matrA.length; i++) {

**for** (**int** j = 0; j < (i-1); j++) {

**if**(matrA[k][j] < matrA[k][j+1]){

temp = matrA[k][j];

matrA[k][j] = matrA[k][j+1];

matrA[k][j+1] = temp;

}

}

}

}

**return** matrA;

}

/\*\*

\* Множить матрицю на число

\* **@param** a число

\* **@param** matrB матриця

\* **@return** результат множення

\*/

**public** **static** **double**[][] multNumbMatr(**double** a, **double** matrB[][]){

**for** (**int** i = 0; i < matrB.length; i++) {

**for** (**int** j = 0; j < matrB[0].length; j++) {

matrB[i][j] = matrB[i][j]\*a;

}

}

**return** matrB;

}

/\*\*

\* Зчитує матрицю

\* **@param** n кількість елементів

\* **@param** message повідомлення для відображення

\* **@return** зчитану матрицю

\*/

**public** **static** **synchronized** **double**[][] readMatr( **int** n, String message){

**double** matr[][] = **new** **double**[n][n];

System.*out*.println(message);

Scanner reader = **new** Scanner(System.*in*);

**for** (**int** i = 0; i < matr.length; i++) {

**for** (**int** j = 0; j < matr.length; j++) {

// matr[i][j] = reader.nextDouble();

matr[i][j] = 1;

}

}

**return** matr;

}

/\*\*

\* Відображає матрицю

\* **@param** matr матриця для відображення

\* **@param** message повідомлення для відображення

\*/

**public** **static** **synchronized** **void** showMatr(**double** matr[][], String message){

System.*out*.println(message);

**for** (**int** i = 0; i < matr.length; i++) {

**for** (**int** j = 0; j < matr[i].length; j++) {

System.*out*.print(matr[i][j] + " ");

}

System.*out*.println();

}

System.*out*.println();

}

/\*\*

\* Зчитує вектор

\* **@param** n кількість елементів вектора

\* **@param** message повідомлення для відображення

\* **@return** зчитаний вектор

\*/

**public** **static** **synchronized** **double**[] readVect(**int** n, String message){

**double** vect[] = **new** **double** [n];

Scanner reader = **new** Scanner(System.*in*);

System.*out*.println(message);

**for** (**int** i = 0; i < vect.length; i++) {

// vect[i] = reader.nextDouble();

vect[i] = 1;

}

**return** vect;

}

/\*\*

\* Відображає вектор

\* **@param** vect вектор для відображення

\* **@param** message повідомлення для відображення

\*/

**public** **static** **synchronized** **void** showVect(**double** vect[], String message){

System.*out*.println(message);

**for** (**int** i = 0; i < vect.length; i++) {

System.*out*.print(vect[i] + " ");

}

System.*out*.println();

}

}