МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра інформаційних систем та мереж



Лабораторна робота №4

з дисципліни

«Технології розподілених систем та паралельних обчислень»  
на тему

**«Застосування процесу декомпозиції в паралельному програмуванні»**

Варіант 25(10)

Виконав:

студент групи КН–47

Шандра О.С.

Прийняв:

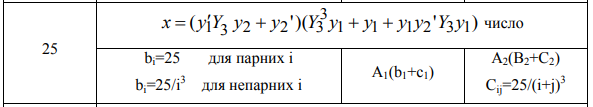
к.т.н., доцент каф. ІСМ

Якушев В.С.

Львів–2019

**Мета роботи:** ознайомитись та вивчити на практиці методи декомпозиції.

**Завдання**



**Хід роботи**

using System;

using System.Text;

namespace Lab4

{

class Program

{

static void Main(string[] args)

{

Matrix m1 = new Matrix

(new int[,] {{1, 3, 2, 8, 1, 3},

{6, 3, 8, 1, 3, 3},

{9, 1, 3, 5, 2, 3},

{2, 6, 9, 2, 9, 3},

{5, 3, 2, 6, 4, 3},

{0, 1, 5, 7, 4, 3}});

var d = new Decomposition();

Console.WriteLine("Результат: " + d.CalculateX(m1));

}

}

class Decomposition

{

private double i { get; set; }

private double j { get; set; }

private double b1 { get { return 25; } }

private double b2 { get { return 25.00 / Math.Pow(i, 3); } }

private double C { get { return 25.00 / Math.Pow(i + j, 3); } }

public double CalculateX(Matrix m)

{

i = 1;

j = 1;

return C;

}

}

/// <summary>

/// Class Matrix that contain

/// values array, it's Width and Height

/// and some static methods

/// </summary>

public class Matrix

{

public Matrix(int[,] array)

{

this.array = array;

}

private int[,] array;

public int Height

{

get

{

return array.GetLength(0);

}

}

public int Width

{

get

{

return array.GetLength(1);

}

}

/// <summary>

/// Getting complementary minor

/// </summary>

/// <param name="M">Matrix search minor</param>

/// <param name="k">Minor order</param>

/// <returns>Double Get minor wiht M and k - 1 parameters</returns>

public static double GetComplementaryMinor(Matrix M, int k)

{

return GetMinor(M, k - 1);

}

/// <summary>

/// Getting k-ordered minor

/// </summary>

/// <param name="M">Matrix search minor</param>

/// <param name="k">Minor order</param>

/// <returns>Double Get minor wiht M, k and k parameters</returns>

public static double GetMinor(Matrix matrix, int k)

{

var maxMinorRange = Math.Min(matrix.Height, matrix.Width);

var square = new int[maxMinorRange, maxMinorRange];

//Getting square from rectengle

for (int i = 0; i < maxMinorRange; i++)

{

for (int j = 0; j < maxMinorRange; j++)

{

square[i, j] = matrix.array[i, j];

}

}

return GetMinor(new Matrix(square), k, k);

}

/// <summary>

/// Getting minor with removing row and collum

/// </summary>

/// <param name="M">Matrix search minor</param>

/// <param name="i">Row removed</param>

/// <param name="j">Collum removed</param>

/// <returns>Dooble Determinant</returns>

public static double GetMinor(Matrix matrix, int i, int j)

{

//In case non-square matrix

if (matrix.Height != matrix.Width)

{

throw new ArgumentException("No i- j- mionors for rectangle matrix");

}

var size = matrix.Width;

//Matrix with removing row i

var matrixWithoutRow = new int[size - 1, size];

//Result matrix - matrix with removing row i and collum j

var resultMatrix = new int[size - 1, size - 1];

//Creating Matrix with removing row i

for (int n = 0; n < size - 1; n++)

{

for (int m = 0; m < size; m++)

{

if (n < i)

{

matrixWithoutRow[n, m] = matrix.array[n, m];

}

else

{

matrixWithoutRow[n, m] = matrix.array[n + 1, m];

}

}

}

//Creating result matrix

for (int n = 0; n < size - 1; n++)

{

for (int m = 0; m < size - 1; m++)

{

if (m < j)

{

resultMatrix[n, m] = matrixWithoutRow[n, m];

}

else

{

resultMatrix[n, m] = matrixWithoutRow[n, m + 1];

}

}

}

return Determinant(new Matrix(resultMatrix));

}

/// <summary>

/// Adding two minors

/// </summary>

/// <param name="minor1">First minor to adding</param>

/// <param name="minor2">Second minor to adding</param>

/// <returns>Result of adding</returns>

public static double AddMinors(double minor1, double minor2)

{

return minor1 + minor2;

}

/// <summary>

/// Multiplication two minors

/// </summary>

/// <param name="minor1">First minor to multiplicate</param>

/// <param name="minor2">Second minor to multiplicate</param>

/// <returns>Result of multiplication</returns>

public static double MultiplicateMinors(double minor1, double minor2)

{

return minor1 \* minor2;

}

/// <summary>

/// Dividing two minors

/// </summary>

/// <param name="minor1">First minor</param>

/// <param name="minor2">Second minor to divide</param>

/// <returns>Result of dividing</returns>

public static double DivideMinors(double minor1, double minor2)

{

return minor1 / minor2;

}

/// <summary>

/// Finding determinant of Matrix

/// </summary>

/// <param name="Matrix">Target Matrix</param>

/// <returns>Double result</returns>

public static double Determinant(Matrix inputSquare)

{

//In case of one-value matrix.

if (inputSquare.Height == 1)

{

return inputSquare.array[0, 0];

}

//In case of two-order matrix.

else if (inputSquare.Height == 2)

{

//Calculating by mathematical formula.

return inputSquare.array[0, 0] \* inputSquare.array[1, 1]

- inputSquare.array[1, 0] \* inputSquare.array[0, 1];

}

double result = 0;

int size = inputSquare.Width;

for (int i = 0; i < size; i++)

{

//Calculating by mathematical formula.

result += Math.Pow(-1, i) \* inputSquare.array[0, i] \* GetMinor(inputSquare, 0, i);

}

return result;

}

/// <summary>

/// Converting to string for convenience presentation format

/// </summary>

/// <return>String look</return>

public override string ToString()

{

StringBuilder result = new StringBuilder();

for (int i = 0; i < Height; i++)

{

result.Append("( ");

for (int j = 0; j < Width; j++)

{

result.Append(array[i, j]);

result.Append(" ");

}

result.Append(")");

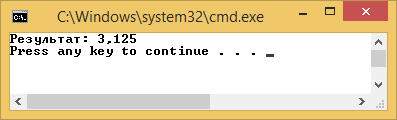
result.AppendLine();

}

return result.ToString();

}

} }



*Рис. 1. Результат виконання програми*

**Висновок**

В результаті виконання даної роботи було ми на практиці ознайомилися та вивчили методи декомпозиції.