Пояснительная записка по микропроекту №1

> ВУЗ: НИУ "Высшая школа экономики"
> Факультет: ФКН
> Образовательная программа: Программная инженерия
> Название разработки: "Домашнее задание №3"
> Студент: Елесин Сергей Владимирович
> Группа: БПИ-193
> Вариант: 5

Задание:

Определить ранг матрицы. Входные данные: целое положительное число n, произвольная матрица A размерности n x n. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

Идея решения:

Из курса алгебры известно, что если в матрице определители всех миноров порядка N равны 0, значит ранг матрицы меньше N. Нулевая матрица имеет ранг 0. Для вычисления ранга заданной матрицы оптимально будет распарраллелить вычисление определителей миноров.

Алгоритм:

1. Считываем все данные из файла
2. Вычисляем определители всех миноров (с разными размерами и со всеми возможными смещениями)
3. Если хотя бы один определитель минора порядка N не равен нулю, помечаем, что ранг матрицы не меньше N.
4. В конце на основе наших пометок определяем ранг матрицы (ищем первое N, при котором все определители миноров данного порядка равны нулю)

Модель вычисления

Для реализации программы была использована модель "Управляющий и рабочие". Основной поток "раздаёт работу" остальным потокам и дожидается их выполнения.
Вспомогательные потоки результат своих вычислений записывают в общую переменную, доступ к которой одновременно может быть только у одного потока.

Формат входных данных

При запуске программы из консоли можно указать в качестве аргумента абсолютный или относительный путь к тестовому файлу (например: "../test1.txt")
В тестовом файле указываются сначала число вспомогательных потоков, затем размер матрицы (N), а после - N^2 элементов матрицы (по строкам). Все числа указываются с новой строки.

Ограничения на входные данные:

- число вспомогательных потоков должно быть не меньше 1
- размер матрицы должен быть в диапазоне от 1 (включительно) до 10 (включительно), так как вычисление определителя матрицы - далеко не самая быстрая операция
- элементы матрицы должны быть целыми числами в диапазоне от -2^{31} до $2^{31} - 1$

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace RangMatrix
{
    class Program
    {
        private static int counter;
        private static int threadAmount;  // количество потоков
        //private static Queue<Task> tasks;      // массив потоков
        private static Matrix matrix;     // исходная матрица
        private static bool[] result;     // массив для вычисления результата
        public static Task[] tasks;
        private static CancellationTokenSource[] cancelSources;
        private static object locker = new object();


        /// <summary>
        /// Метод, для считывания информации из файла
        /// </summary>
        /// <param name="filePath"> Путь до файла </param>
        static void ReadFromFile(string filePath)
        {
            int[,] array;

            using (StreamReader sr = new StreamReader(filePath))
            {
                threadAmount = int.Parse(sr.ReadLine());
                int size = int.Parse(sr.ReadLine());

                if (threadAmount < 1 || size < 1)
                    throw new ArgumentException("Число потоков и размер массива не могут быть м

                array = new int[size, size];

                for (int y = 0; y < size; ++y)
                    for (int x = 0; x < size; ++x)
                        array[y, x] = int.Parse(sr.ReadLine());
            }
            matrix = new Matrix(array);
        }


        /// <summary>
        /// Точка входа в программу
        /// </summary>
        /// <param name="args"> Аргументы, передаваемые через консоль </param>
        static void Main(string[] args)
        {
```

```
        try
        {
            ReadFromFile(args.Length > 0 ? args[0] : "test1.txt"); // считываем файл
            result = new bool[matrix.size + 1]; // инициализируем массив
            result[0] = !matrix.IsZero(); // определяем, является ли матрица нулевой

            CalculateRang(matrix); // запускаем вычисление ранга на нескольких потоках

            // выводим результат в консоль
            for (int i = 0; i < result.Length; ++i)
                if (!result[i])
                {
                    Console.WriteLine("Ранг матрицы = " + i);
                    break;
                }
        }
        catch (IOException io)
        {
            Console.WriteLine("Во время чтения из файла произошла ошибка, " +
                              "убедитесь в исправности файла с тестовыми данными");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Во время выполнения произошла ошибка: " + ex.Message);
        }

        Console.WriteLine("\nНажмите любую клавишу для завершения программы...");
        Console.ReadKey(true);
    }


    /// <summary>
    /// Метод, который будут выполнять потоки
    /// </summary>
    /// <param name="minorSize"> Размер искомого минора </param>
    /// <param name="offsetX"> Смещение по горизонатали </param>
    /// <param name="offsetY"> Смещение по вертикали </param>
    static void Work(int minorSize, int offsetX, int offsetY)
    {
        if (result[minorSize - 1] ||
            Matrix.Determinant(Matrix.Minor(matrix.array, minorSize, offsetX, offsetY)) ==
            return;

        lock (locker)  // доступ к переменной result предоставляется одновременно лишь одно
        {
            result[minorSize - 1] = true;
        }
    }

    /// <summary>
    /// Вычисляем ранг матрицы
    /// </summary>
    static void CalculateRang(Matrix matrix)
    {
        // инициализируем массив потоков
```

```
            tasks = new Task[threadAmount];
            for (int i = 0; i < threadAmount; ++i)
                tasks[i] = Task.FromResult(0);

            // Вычисляем определители всех миноров данной матрицы
            for (int s = 2; s <= matrix.size; ++s)
                for (int y = 0; y <= matrix.size - s; ++y)
                    for (int x = 0; x <= matrix.size - s; ++x)
                    {
                        // создаём локальные копии изменяемых переменных
                        int _s = s, _x = x, _y = y;

                        // запускаем выполнение метода на освободившемся потоке
                        tasks[Task.WaitAny(tasks)] = Task.Run(() => Work(_s, _x, _y));
                    }
            // дожидаемся выполнения всех потоков
            Task.WaitAll(tasks);
        }
    }


    /// <summary>
    /// Класс матрицы
    /// </summary>
    public class Matrix
    {
        public readonly int[,] array;
        public int size => array.GetLength(0);
        public bool IsZero() => array.Cast<int>().All(value => value == 0);

        // конструктор
        public Matrix(int[,] array) => this.array = array;

        /// <summary>
        /// Вычисление определителя матрицы
        /// </summary>
        /// <param name="array"> матрица </param>
        /// <returns> Определитель матрицы </returns>
        public static int Determinant(int[,] array)
        {
            int n = (int)Math.Sqrt(array.Length);

            if (n == 1)
                return array[0, 0];

            int det = 0;

            for (int k = 0; k < n; k++)
                det += array[0, k] * Cofactor(array, 0, k);

            return det;
        }


        private static int Cofactor(int[,] array, int row, int column) =>
```

```
        (int)Math.Pow(-1, column + row) * Determinant(Minor(array, row, column));

        /// <summary>
        /// Нахождение минора матрицы указанного размера и со смещениями по горизонатали и верт
        /// </summary>
        /// <returns> Минор матрицы </returns>
        public static int[,] Minor(int[,] array, int minorSize, int offsetX, int offsetY)
        {
            int[,] minor = new int[minorSize, minorSize];

            for (int y = 0; y < minorSize; ++y)
                for (int x = 0; x < minorSize; ++x)
                    minor[y, x] = array[offsetY + y, offsetX + x];

            return minor;
        }

        /// <summary>
        /// Нахождение минора матрицы за вычетом указанных столбца и ряда
        /// </summary>
        /// <returns> Минор матрицы </returns>
        private static int[,] Minor(int[,] array, int row, int column)
        {
            int n = (int)Math.Sqrt(array.Length);
            int[,] minor = new int[n - 1, n - 1];

            int _i = 0;
            for (int i = 0; i < n; i++)
            {
                if (i == row) continue;

                int _j = 0;
                for (int j = 0; j < n; j++)
                {
                    if (j == column) continue;

                    minor[_i, _j] = array[i, j];
                    _j++;
                }
                _i++;
            }
            return minor;
        }
    }
}
```

Тесты:

- test0.txt - проверка работоспособности программы (правильный ответ: 2)
- test1.txt - проверка работы при наличии линейно-зависимых строк в матрице (правильный ответ: 2)

- test2.txt - дана нулевая матрица (правильный ответ: 0)
- test3.txt - задано неверное (отрицательное) значение для одной из переменных (правильный ответ: выдача ошибки с указанием причины)
- test4.txt - проверка работы при условии, что все строки в матрице - линейно-зависимые (правильный ответ: 1)
- test5.txt - проверка работы программы при максимальном размере матрицы (правильный ответ: 10)

Источники:

http://www.mathprofi.ru/rang_matricy.html - теория о рангах матрицы
https://habr.com/ru/post/452094/ - работа с многопоточностью в .NET
https://metanit.com/sharp/tutorial/11.4.php - синхронизация потоков (lock)
https://metanit.com/sharp/tutorial/12.2.php - работа с классом Task
https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task.waitany?view=netcore-3.1 - метод WaitAny()
https://pro-prof.com/forums/topic/parallel-programming-paradigms - парадигмы параллельного программирования