Министерство науки и высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №6
«Исследование способов профилирования программного обеспечения»
по дисциплине
«ТЕСТИРОВАНИЕ ПО»

Выполнил студент группы ИС/б-17-2-о
Горбенко К. Н.
Проверил
Тлуховская Н.П.

Севастополь
2019

1 ЦЕЛЬ РАБОТЫ

Исследовать критические по времени выполнения участки программного кода и возможности их устранения. Приобрести практические навыки анализа программ с помощью профайлера EQATECProfiler.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Разработать программу на основе библиотеки классов, реализованной и протестированной в предыдущей работе. Программа должна как можно более полно использовать функциональность класса. При необходимости для наглядности профилирования в методы класса следует искусственно внести задержку выполнения.

2. Выполнить профилирование разработанной программы, выявить функции, на выполнение которых тратится наибольшее время.

3. Модифицировать программу с целью оптимизации времени выполнения.

4. Выполнить повторное профилирование программы, сравнить новые результаты и полученные ранее, сделать выводы.

3 ХОД РАБОТЫ

Для тестирования напишем следующую программу:

```
1 public static void Main(string[] args)
2 {
3     var matrix = CreateMatrix();
4     var sumsOfPositiveElementsInColumns = MatrixOperations.
          GetSumsOfPositiveElementsInColumns(matrix);
5
6     foreach (var item in sumsOfPositiveElementsInColumns)
7     {
8         Console.WriteLine($"{item} ");
9     }
10    return;
11 }
12
13 private static int[,] CreateMatrix()
14 {
15    var random = new Random();
16    var matrix = new int[10000, 10000];
17
18    for (var i = 0; i < 10000; i++)
```

```
19      {
20          for (var j = 0; j < 10000; j++)
21          {
22              matrix[i, j] = Enumerable.Range(0, random.Next(1, 1000)).Select(x
                    => random.Next(int.MinValue, int.MaxValue))
23                                          .ToArray()
24                                          .Last();
25          }
26      }
27
28      return matrix;
29  }
```

В данной программе создается массив 10000x10000 целочисленных элементов. Каждый элемент создается путем взятия последнего элемента последовательности случайных чисел случайной длины. Затем, для каждого столбца матрицы рассчитывается сумма положительных элементов. В конце каждый элемент преобразовывается к строке, из каждой части склеивается одна строка и выводится на экран.

Потенциально, наибольшее время может тратиться на создание массива, вычисление суммы столбцов, преобразование элементов к строкам, запись большой строки в консоль. Запустим профилировщик:

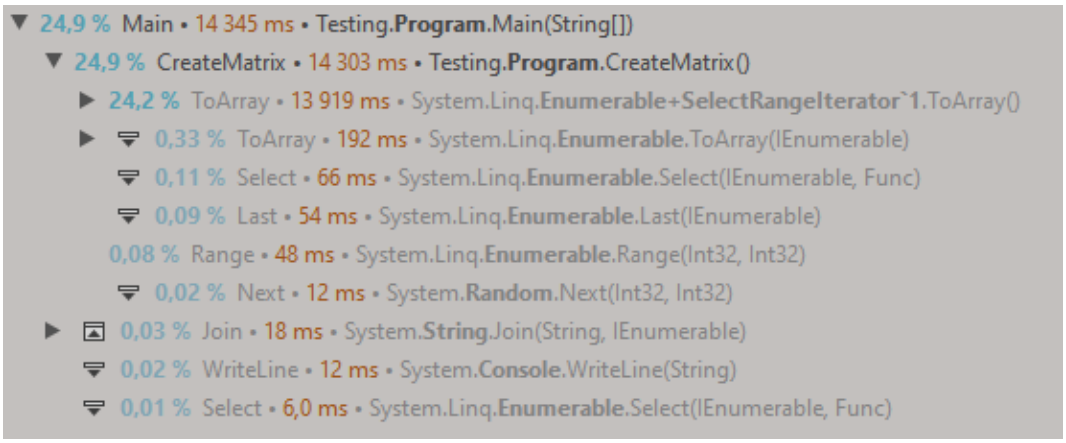

Рисунок 1 – Результаты профилирования программы

Время выполнения программы - 14,345 с. Из профилирования программы видно, что подавляющее большинство времени программы заняли вызовы метода ToArray(). При вызове этого метода происходило копирование элементов в новый массив. Исправим это место:

```
1 public static void Main(string[] args)
```

```
{
    var matrix = CreateMatrix();
    var sumsOfPositiveElementsInColumns = MatrixOperations.
        GetSumsOfPositiveElementsInColumns(matrix);

    foreach (var item in sumsOfPositiveElementsInColumns)
    {
        Console.WriteLine($"{item} ");
    }
    return;
}

private static int[,] CreateMatrix()
{
    var random = new Random();
    var matrix = new int[10000, 10000];

    for (var i = 0; i < 10000; i++)
    {
        for (var j = 0; j < 10000; j++)
        {
            matrix[i, j] = Enumerable.Range(0, random.Next(1, 1000)).Select(x
                => random.Next(int.MinValue, int.MaxValue))
                                    .Last();
        }
    }

    return matrix;
}
```

Снова запустим профилировщик:

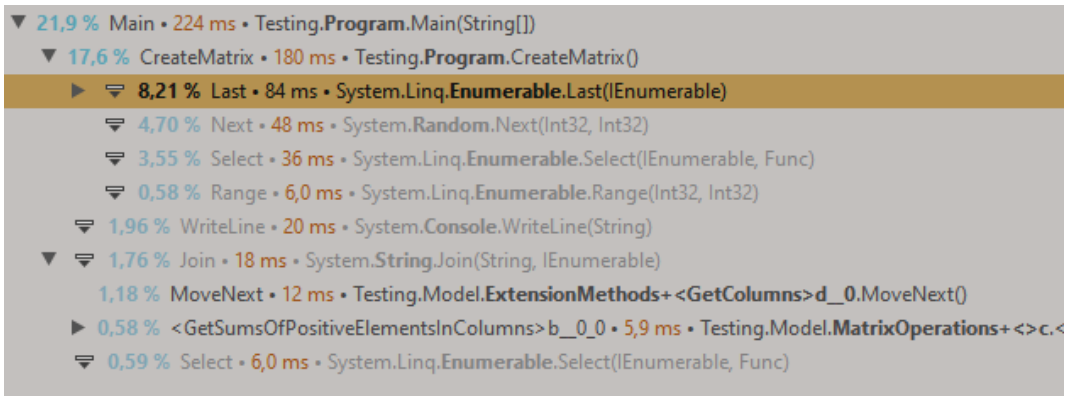

Рисунок 2 – Результаты профилирования программы

Время выполнения программы сократилось 0.224 с.

ВЫВОДЫ

В ходе лабораторной работы было выполнено профилирование программы. Оно позволяет оценить время и память, затрачиваемые на вызов каждой функции. Это позволяет определить места в программном коде, на которые стоит обратить внимание. Кроме того, профилировщик позволяет построить граф вызовов функций. Это полезно например, для того, чтобы определить частоту вызовов той или иной функции.