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

```
19
           for (var j = 0; j < 10000; j++)
20
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 10000х10000 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:

```
    ▼ 24,9 % Main • 14 345 ms • Testing.Program.Main(String[])
    ▼ 24,9 % CreateMatrix • 14 303 ms • Testing.Program.CreateMatrix()
    ▶ 24,2 % ToArray • 13 919 ms • System.Linq.Enumerable+SelectRangeIterator`1.ToArray()
    ▶ ♥ 0,33 % ToArray • 192 ms • System.Linq.Enumerable.ToArray(IEnumerable)
    ♥ 0,11 % Select • 66 ms • System.Linq.Enumerable.Select(IEnumerable, Func)
    ♥ 0,09 % Last • 54 ms • System.Linq.Enumerable.Last(IEnumerable)
    0,08 % Range • 48 ms • System.Linq.Enumerable.Range(Int32, Int32)
    ♥ 0,02 % Next • 12 ms • System.Random.Next(Int32, Int32)
    ▶ ☑ 0,03 % Join • 18 ms • System.String.Join(String, IEnumerable)
    ♥ 0,02 % WriteLine • 12 ms • System.Console.WriteLine(String)
    ♥ 0,01 % Select • 6,0 ms • System.Linq.Enumerable.Select(IEnumerable, Func)
```

Рисунок 1 – Результаты профилирования программы

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

```
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()
15
      var random = new Random();
      var matrix = new int[10000, 10000];
16
17
18
      for (var i = 0; i < 10000; i++)
19
           for (var j = 0; j < 10000; j++)
20
21
           {
22
               matrix[i, j] = Enumerable.Range(0, random.Next(1, 1000)).Select(x
                  => random.Next(int.MinValue, int.MaxValue))
23
                                          .Last();
24
           }
25
26
27
      return matrix;
28 }
```

#### Снова запустим профилировщик:

```
    ▼ 21,9 % Main • 224 ms • Testing.Program.Main(String[])
    ▼ 17,6 % CreateMatrix • 180 ms • Testing.Program.CreateMatrix()
    ▶ ▼ 8,21 % Last • 84 ms • System.Linq.Enumerable.Last(IEnumerable)
    ▼ 4,70 % Next • 48 ms • System.Random.Next(Int32, Int32)
    ▼ 3,55 % Select • 36 ms • System.Linq.Enumerable.Select(IEnumerable, Func)
    ▼ 0,58 % Range • 6,0 ms • System.Linq.Enumerable.Range(Int32, Int32)
    ▼ 1,96 % WriteLine • 20 ms • System.Console.WriteLine(String)
    ▼ 1,76 % Join • 18 ms • System.String.Join(String, IEnumerable)
    1,18 % MoveNext • 12 ms • Testing.Model.ExtensionMethods + <GetColumns>d_0.MoveNext()
    ▶ 0,58 % <GetSumsOfPositiveElementsInColumns>b_0_0 • 5,9 ms • Testing.Model.MatrixOperations + <>c.
    ▼ 0,59 % Select • 6,0 ms • System.Linq.Enumerable.Select(IEnumerable, Func)
```

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.