Лабораторная работа №2

Ибраев Ерлан Иржанович 184-1 МОиАИС

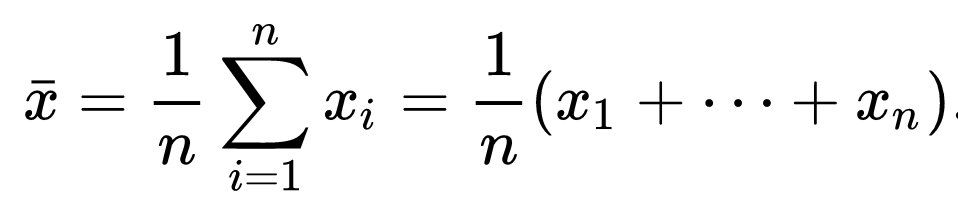
Цель работы: оценить скорость выполнения операция в памяти с разным порядком доступа

Ответы на вопросы:

1. Доступ к данным в кэше осуществляется быстрее, после первого заполнения в кэше
2. Все в проблеме доступа памяти. Если смотреть слева направо способ заполнения мы видим, что скорость заполнение выше, т. к. он заполняет следующий элемент ячейки. Но если идти по проходу справа налево, то тут получаем большее затрачиваемого времени на получение доступа памяти ячейки.
3. Время на доступ к памяти существенные выше, потому что идет поиск на нахождение заданного массива
4. **Отчет по блоку 1:**

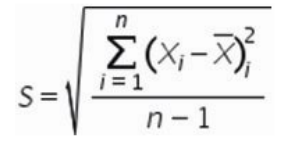
Ссылки на источники указаны ниже. Картинки формул взяты от туда же. Код метод

Среднее – арифметическое среднее значений.



static double Average(double[] output)  
{  
 double sum = 0;  
 for(int i = 0; i < output.Length; i++)   
 sum += output[i];  
 return sum / output.Length;  
}

Дисперсия – среднее значение квадратов отклонений от среднего.

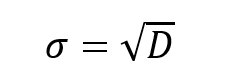


static double Dispersion(double[] output)  
{  
 double average = Average(out);  
 double sum = 0;  
 for (int i = 0; i < output.Length; i++)  
 sum += Math.Pow(output[i] - average, 2);  
 return sum / output.Length;  
}

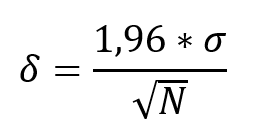
public static double StandardDeviation(double[] output)  
{  
 double dispersion = Dispersion(output);  
 return Math.Sqrt(dispersion);  
 }

public static double Median(double[] output)  
{  
 if (output.Length % 2 == 0)  
 return (output[output.Length / 2] + output[output.Length / 2 - 1]) / 2;  
 else  
 return output[output.Length / 2 + 1];  
}

Среднеквадратическое отклонение – корень из дисперсии.



На основе полученных данных определите интервальную оценку для матожидания времени работы алгоритма с 95% уровнем значимости.

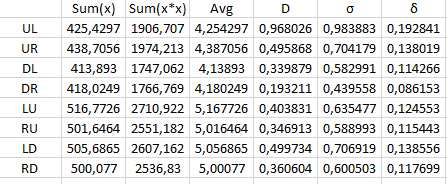


static string ConfidenceInterval(double[] output)

{  
 double average = Average(output);  
 double standartdeviation = StandardDeviation(output);  
 double standarterror = 2 \* standartdeviation / Math.Sqrt(output.Length - 1);  
 return ("(" + (average - standarterror).ToString() + ":" + (average + standarterror).ToString() + ")");

}

Сводная таблица полученных значений по формулам.



Данные циклов по первому блоку:

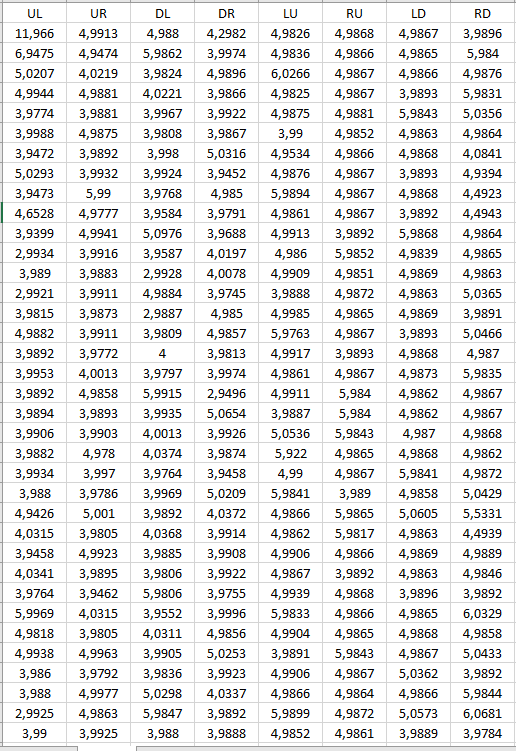
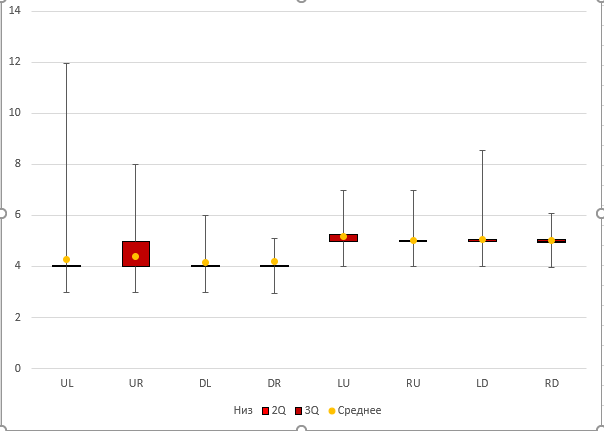


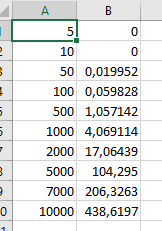
Диаграмма ящик с усами



Вывод: первые четыре метода, намного лучше время, чем у других. Потому что они способ заполнения массива у всех разный. Вся проблема получение доступа памяти на определенную ячейку. Тогда, когда у первых четырех идет последовательная получение доступ памяти.

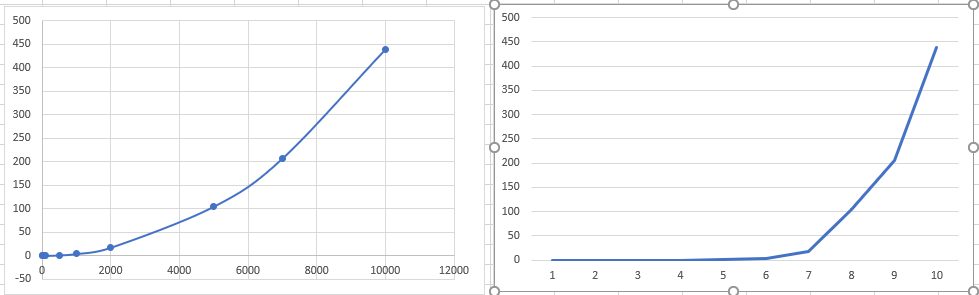
5)Отчет по Блоку 2

Самое лучшее время у UL.



Скриншот результатов программы по заполнению массива данных.

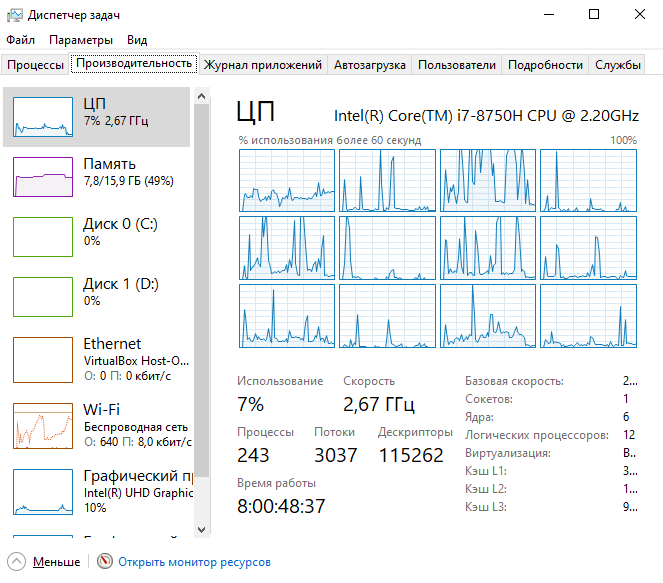
График по исходным данным:



Вывод: по первому графику линию более точнее показывает линию по заданным значениям.

**6) Скриншоты программы**

Работа была выполнена на консоли, есть скрин нагрузки на процессор. (Вдруг понадобится).



**7) Список источник**

<https://exceltip.ru/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B-%D1%8F%D1%89%D0%B8%D0%BA-%D1%81-%D1%83%D1%81%D0%B0%D0%BC%D0%B8-%D0%B2-excel/>

<https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D0%B8%D0%B0%D0%BD%D0%B0_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0)>

<https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%BD%D0%B5%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D1%82%D0%BA%D0%BB%D0%BE%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5>

<http://www.machinelearning.ru/wiki/index.php?title=Регрессионный_анализ>

<https://mathprofi.net/formula_dispersii_standartnoe_otklonenie_koefficient_variacii.html>

<http://www.mathprofi.ru/statisticheskie_ocenki_parametrov_generalnoy_sovokupnosti.html>

**8) Код программы**

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;

namespace ConsoleApp2  
{  
 class Program  
 {  
 delegate double GetTimeOfMethodCycle(double[,] matrix, int N);  
 static void Main(string[] args)  
 {  
 BlockTask1();  
 BlockTask2();  
 }  
 static void BlockTask1()  
 {  
 string[] output = new string[101];//Для фиксирование данных  
 output[0] += "UL;UR;DL;DR;LU;RU;LD;RD";//Для первой строчки в Excel файле  
 int K = 100, N = 1000;//Количество циклов  
 double[,] matrix = new double[N, N];  
 GetTimeOfMethodCycle SecectCyrcly;//Выбор работы циклов  
 GetTimeOfMethodCycle[] TempArray = new GetTimeOfMethodCycle[8] { UL, UR, DL, DR, LU, RU, LD, RD };//Массив для перебора циклов

for (int d = 1; d <= 8; d++)

{

for (int o = 1; o <= K; o++)

{

SecectCyrcly = TempArray[d - 1];

double time = SecectCyrcly(matrix, N);//Вызываем выбранный цикл

output[o] += time.ToString()+";";//Фиксирует время работы

}

File.WriteAllLines(@"..\..\OutputBlock1.csv", output, Encoding.Default);//Запись данных в файл

}

}

static void BlockTask2()//Блок 2

{

string[] output = new string[10];//Для фиксирование данных

int[] ArrayChecks = new int[] { 5, 10, 50, 100, 500, 1000, 2000, 5000, 7000, 10000 };//Количество размеров матрицы

int K = 50;//Количество повторений цикла

for (int i = 0; i < ArrayChecks.Length; i++) //Проход по всем матрицам

{

double dt = 0;//Фиксирование времени

double[,] ArrSecectCheck = new double[ArrayChecks[i], ArrayChecks[i]];//Чтобы найти усреднее значение времени

for (int repeat = 0; repeat < K; repeat++)

{

dt += UL(ArrSecectCheck, ArrayChecks[i]);

}

dt /= K;//Находим среднее время

output[i] += ($"{ArrayChecks[i].ToString()}; {dt.ToString()};");

File.WriteAllLines(@"..\..\OutputBlock2.csv", output, Encoding.Default);//Записываем данные в файл Excel

}

}

//Циклы

static double UL(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

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

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

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double UR(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

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

for (int j = N - 1; j >= 0; j--)

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double DL(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

for (int i = N - 1; i >= 0; i--)

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

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double DR(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

for (int i = N - 1; i >= 0; i--)

for (int j = N - 1; j >= 0; j--)

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double LU(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

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

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

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double RU(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

for (int j = N - 1; j >= 0; j--)

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

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double LD(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

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

for (int i = N - 1; i >= 0; i--)

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

static double RD(double[,] matrix, int N)

{

DateTime t0 = DateTime.Now;

for (int j = N - 1; j >= 0; j--)

for (int i = N - 1; i >= 0; i--)

matrix[i, j] = i / (j + 1.0);

return (DateTime.Now - t0).TotalMilliseconds;

}

}

}