Государственное образовательное учреждение высшего профессионального

образования

«Московский государственный технический университет

имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №3

Монахов Д. И.

ИУ7-54

Москва 2018

Оглавление

[1. Аналитическая часть 3](#_Toc527818313)

[1.1. Определение произведения матриц 3](#_Toc527818314)

[2. Конструкторская часть 4](#_Toc527818315)

[2.1 Разработка алгоритмов 4](#_Toc527818316)

[3. Технологическая часть 7](#_Toc527818317)

[3.1 Требования к программному обеспечению 7](#_Toc527818318)

[3.2 Средства реализации 7](#_Toc527818319)

[3.3 Листинг кода 8](#_Toc527818320)

[4. Экспериментальная часть 11](#_Toc527818321)

[4.1 Примеры работы 11](#_Toc527818322)

[4.2 Постановка эксперимента 11](#_Toc527818323)

[Заключение 12](#_Toc527818324)

# 1. Аналитическая часть

В этой части рассматривается, что такое перемножение матриц, рассматриваются простой алгоритм перемножения матриц и алгоритм Винограда.

## 1.1. Определение произведения матриц

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Пусть даны две матрицы: A с размерами l x m и B с размерами m x n. Произведением матрицы A на B будет матрица C, элементы которой можно найти по формуле (1).

# 2. Конструкторская часть

В этой части будут составлен псевдокод алгоритмов, а сами алгоритмы будут подвергнуты сравнению аналитически.

## 2.1 Разработка алгоритмов

Псевдокод простого алгоритма умножения матриц

Функция ПростоеУмножение (Матрица1, Матрица2); Возвращает Матрицу

Начало

Для I от 0 до количества столцов в Матрице1

Для J от 0 до количество строк в Матрице2

R[i, j] = 0

Для K от 0 до количества строк в Метрице1

R[i, j] = R[i, j] + Матрица1[i, k] \* Матрица2[k, j]

КЦ

КЦ

КЦ

Вернуть R

Конец

Псевдокод алгоритма Винограда

d = b/2

// вычисление rowFactors для G

for i = 1 to a do

rowFactor[i] = G[i, 1] \* G[i, 2]

for j = 2 to d do

rowFactor[i] = rowFactor[i] + G[i, 2j — 1] \* G[i, 2j]

end for j

end for i

// вычисление columnFactors для H

for i = 1 to c do

columnFactor[i] = H[1, i] \* H[2, i]

for j = 2 to d do

columnFactor[i] = columnFactor[i] + H[2j — 1, i] \* H[2j, i]

end for j

end for i

// вычисление матрицы R

for i = 1 to a do

for j = 1 to c do

R[i, j] = -rowFactor[i] — columnFactor[j]

for k = 1 to d do

R[i, j]=R[i, j]+(G[i, 2k-1]+H[2k, j])\*(G[i, 2k] + H[2k-1, j])

end for k

end for j

end for i

// прибавление членов в случае нечетной общей размерности

if (2 \* (b/2) /= b) then

for i = 1 to a do

for j = 1 to c do

R[i, j] = R[i, j] + G[i, b] \* H[b, j]

end for j

end for i

end if

Псевдокод улучшенного алгоритма Винограда

d = b/2

for i = 1 to a do

rowFactor[i] = G[i, 1] \* G[i, 2]

for j = 2 to d do

dj = 2 \* j

rowFactor[i] += G[i, dj — 1] \* G[i, dj]

end for j

end for i

for i = 1 to c do

columnFactor[i] = H[1, i] \* H[2, i]

for j = 2 to d do

dj = 2 \* j

columnFactor[i] += H[dj — 1, i] \* H[dj, i]

end for j

end for i

for i = 1 to a do

for j = 1 to c do

R[i, j] = -rowFactor[i] — columnFactor[j]

for k = 1 to d do

dk = 2 \* k

R[i, j] +=(G[i, dk-1]+H[2k, j])\*(G[i, dk] + H[dk-1, j])

end for k

end for j

end for i

if not (b – d = d) then

for i = 1 to a do

for j = 1 to c do

R[i, j] += G[i, b] \* H[b, j]

end for j

end for i

end if

2.2 Оценка сложности алгоритмов.

Введем следующую модель вычислений:

1. Операция сложения
2. Операция умножения
3. Операция деления
4. Операция инверсии знака
5. Операция сравнения (<, >, =)
6. Операция чтения из памяти
7. Операция записи в память
8. Инкремент

Алгоритмическая сложность простого алгоритма

private Matrix Process(Matrix a, Matrix b)

{

Matrix r = new Matrix(a.Cols, b.Rows, \_matrixBuffer);

// i = 0 - запись в память

for (int i = 0; i < r.Rows; i++) // При каждой итерации: чтение, сравнение, инкремент (2)

{

// запись в память

for (int j = 0; j < r.Cols; j++) // При каждой итерации: чтение, сравнение, инкремент (2)

{

r[i, j] = 0; // Запись в память

// Запись в память

for (int k = 0; k < a.Rows; k++) // При каждой итерации: чтение, сравнение, инкремент (2)

{

r[i, j] = r[i, j] + a[i, k] \* b[k, j]; // При каждой итерации: чтение, чтение, чтение, умножение, сложение, запись (6)

}

}

}

return r; // Запись в память

}

Итого: .

В O() нотации эта формула будет записана в виде O()

Алгоритмическая сложность алгоритма Винограда и улучшенного алгоритма Винограда находятся аналогично и составляют приблизительно

# 3. Технологическая часть

В этой части обосновывается выбор языка программирования, необходимых библиотек и будет приведен листинг кодов каждого из алгоритмов.

## 3.1 Требования к программному обеспечению

Требуется вводить две строки и выводить матрицу и значения расстояний, полученных различными реализациями. Допускается предоставление двух режимов ПО: для единичного эксперимента и для массовых экспериментов.

## 3.2 Средства реализации

В качестве языка программирования был выбран C#. Его выбор основан на следующих причинах:

1. язык программирования C# является частью платформы .NET Framework, которая хорошо документирована, обладает хорошей стабильностью работы, и имеет богатый набор библиотечных решений;
2. C# позволяет программисту уделять больше времени написанию кода и не тратить время на побочные действия;
3. C# просто отлаживать, он не обладает уязвимостями языка C++, например, проблемами с указателями;
4. C# по скорости работы почти не уступает языку C++.

Время выполнения замеряется средством .NET под названием StopWatch;

## 3.3 Листинг кода

Листинг простого алгоритма умножения матриц

private Matrix Process(Matrix a, Matrix b)

{

Matrix r = new Matrix(a.Cols, b.Rows, \_matrixBuffer);

for (int i = 0; i < r.Rows; i++)

{

for (int j = 0; j < r.Cols; j++)

{

r[i, j] = 0;

for (int k = 0; k < a.Rows; k++)

{

r[i, j] += a[i, k] \* b[k, j];

}

}

}

return r;

}

Листинг алгоритма Винограда умножения матриц

private Matrix Process(Matrix g, Matrix h)

{

int a = g.Rows;

int b = g.Cols;

int c = h.Cols;

int d = b / 2;

int[] rowFactors = new int[a];

int[] colFactors = new int[c];

Matrix r = new Matrix(a, c);

// Вычисление коэффициентов строк для первой матрицы

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

{

rowFactors[i] = g[i, 0] \* g[i, 1];

for (int j = 2; j <= d; j++)

{

rowFactors[i] = rowFactors[i] + g[i, 2 \* j - 2] \* g[i, 2 \* j - 1];

}

}

// Вычисление коэффициентов столбцов для второй матрицы

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

{

colFactors[i] = h[0, i] \* h[1, i];

for (int j = 2; j <= d; j++)

{

colFactors[i] = colFactors[i] + h[2 \* j - 2, i] \* h[2 \* j - 1, i];

}

}

// Вычисление матрицы R

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

{

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

{

r[i, j] = -rowFactors[i] - colFactors[j];

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

{

r[i, j] = r[i, j] +

(g[i, 2 \* k - 2] + h[2 \* k - 1, j]) \*

(g[i, 2 \* k - 1] + h[2 \* k - 2, j]);

}

}

}

// Прибавление членов в случае нечетной общей размерности

if (2 \* (b / 2) != b)

{

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

{

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

{

r[i, j] = r[i, j] + g[i, b - 1] \* h[b - 1, j];

}

}

}

return r;

}

Листинг улучшенного алгоритма умножения матриц

private Matrix Process(Matrix g, Matrix h)

{

int a = g.Rows;

int b = g.Cols;

int c = h.Cols;

int d = b / 2;

int remainderPart = b - d;

int[] rowFactors, colFactors;

if (\_buffer1 != null && \_buffer1.Length >= a)

{

rowFactors = \_buffer1;

}

else

{

rowFactors = new int[a];

\_buffer1 = rowFactors;

}

if (\_buffer2 != null && \_buffer2.Length >= c)

colFactors = \_buffer2;

else

{

colFactors = new int[c];

\_buffer2 = colFactors;

}

//ClearBuffers();

Matrix r = new Matrix(a, c, \_matrixBuffer);

// Вычисление коэффициентов строк для первой матрицы

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

{

rowFactors[i] = g[i, 0] \* g[i, 1];

for (int j = 2; j <= d; j++)

{

int dj = 2 \* j;

rowFactors[i] += g[i, dj - 2] \* g[i, dj - 1];

}

}

// Вычисление коэффициентов столбцов для второй матрицы

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

{

colFactors[i] = h[0, i] \* h[1, i];

for (int j = 2; j <= d; j++)

{

int dj = 2 \* j;

colFactors[i] += h[dj - 2, i] \* h[dj - 1, i];

}

}

// Вычисление матрицы R

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

{

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

{

r[i, j] = -rowFactors[i] - colFactors[j];

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

{

int dk = 2 \* k;

int dkSub1 = dk - 1;

int dkSub2 = dk - 2;

r[i, j] += (g[i, dkSub2] + h[dkSub1, j]) \*

(g[i, dkSub1] + h[dkSub2, j]);

}

}

}

// Прибавление членов в случае нечетной общей размерности

if (remainderPart != d)

{

int bSub1 = b - 1;

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

{

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

{

r[i, j] += g[i, bSub1] \* h[bSub1, j];

}

}

}

return r;

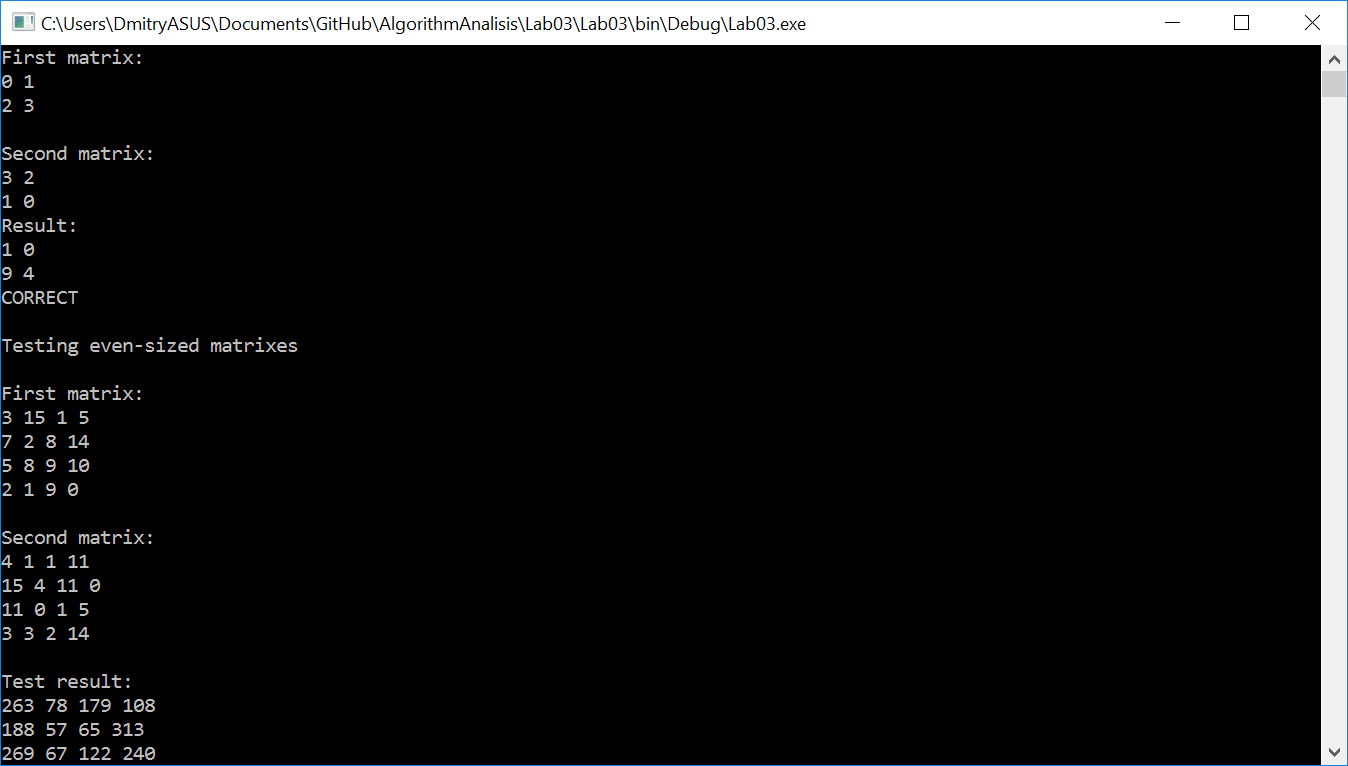
}

# 4. Экспериментальная часть

В этой части лабораторной работы будут поставлены эксперименты над реализованными алгоритмами и сделаны выводы.

## 4.1 Примеры работы

Пример работы программы



## 4.2 Постановка эксперимента

Поставим эксперимент, в котором будет исследовано быстродействие всех трёх алгоритмов. Для этого запустим алгоритмы на матрицах с размерами от 100 до 400 с шагом 100. Чтобы замеренное время было точнее, для каждой матрицы алгоритм будет отработан 100 раз.

Из рисунка 1 видно, что время работы простого алгоритма растет значительно быстрее алгоритма Винограда и его улучшенной версии. Разница во времени работы алгоритмов Винограда и Улучшенного алгоритма Винограда не велика.

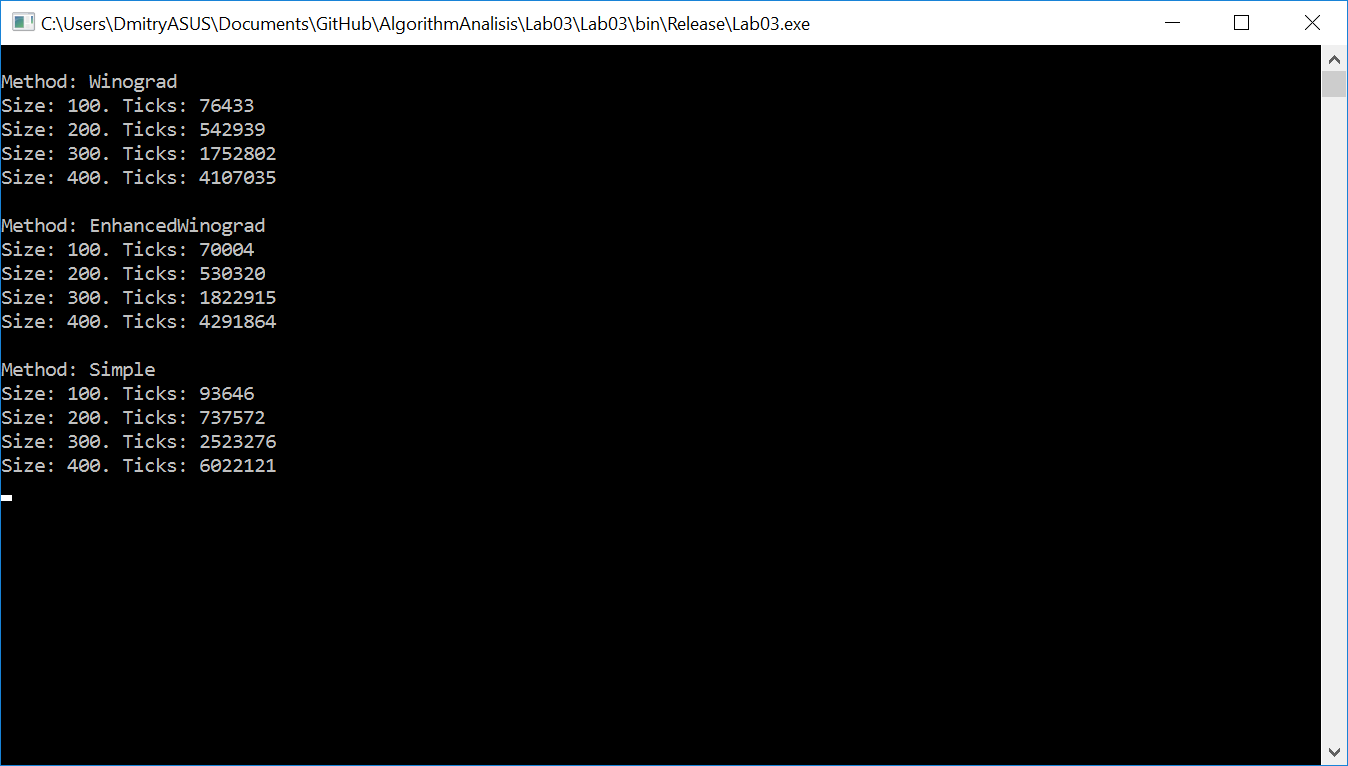


Рисунок 1. Результаты эксперимента

# Заключение

Реализованы алгоритмы умножения матриц. Выяснено экспериментально, что алгоритм Винограда имеет алгоритмическую сложность меньше, чем у простого алгоритма умножения матриц, а следовательно, на матрицах большой размерности алгоритм Винограда является более эффективным по времени.