Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №2
по дисциплине "Анализ Алгоритмов"

Тема Умножение матриц

Студент Светличная А.А.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л., Строганов Ю.В.

Москва — 2022 г.

Оглавление

Введение

Сегодня термин «матрица» применяется во множестве разных областей: от программирования до кинематографии. Матрица в математике – это таблица чисел, состоящая из определенного количества строк и столбцов. Операции обработки матриц, а в частности их умножение, является одной из самых фундаментальных операций в современных вычислениях. Умножение двух матриц возможно, если число столбцов первой матрицы совпадает с числом строк во второй. Результирующая матрица будет иметь столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

1 Аналитическая часть

1.1 Цель и задачи

Целью данной работой является изучение и реализация алгоритмов умножения матриц (стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда), вычисление их трудоемкости. Для достижения поставленной цели требуется решить следующие задачи:

1) изучить и рассмотреть стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;
2) построить блок-схемы данных алгоритмов;
3) реализовать каждый из алгоритмов;
4) оценить трудоемкость алгоритмов;
5) экспериментально оценить временные характеристики алгоритмов;
6) сделать вывод на основании проделанной работы.

1.2 Стандартный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы:

$$A_{mk} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mk} \end{pmatrix}, \quad B_{kn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \dots & b_{kn} \end{pmatrix}. \tag{1.1}$$

Тогда матрица C:

$$C_{mn} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{k} a_{ir} b_{rj} \quad (i = \overline{1,m}; j = \overline{1,n}), \tag{1.3}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.3 Алгоритм Копперсмита-Винограда

Алгоритм Копперсмита-Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита — Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4).

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \tag{1.4}$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм работает быстрее стандартного.

1.4 Оптимизированный алгоритм Копперсмита-Винограда

Оптимизации алгоритма:

1) операции вида $x = x + k$ заменены на $x+ = k$;

2) умножение 2 заменено побитовым сдвигом;

3) происходит предвычисление некоторых слагаемых алгоритма.

Вывод

Были рассмотрены стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Основное отличие алгоритмов Винограда от стандартного алгоритма — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

2.1 Описания алгоритмов

На рисунках ниже показаны схемы стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

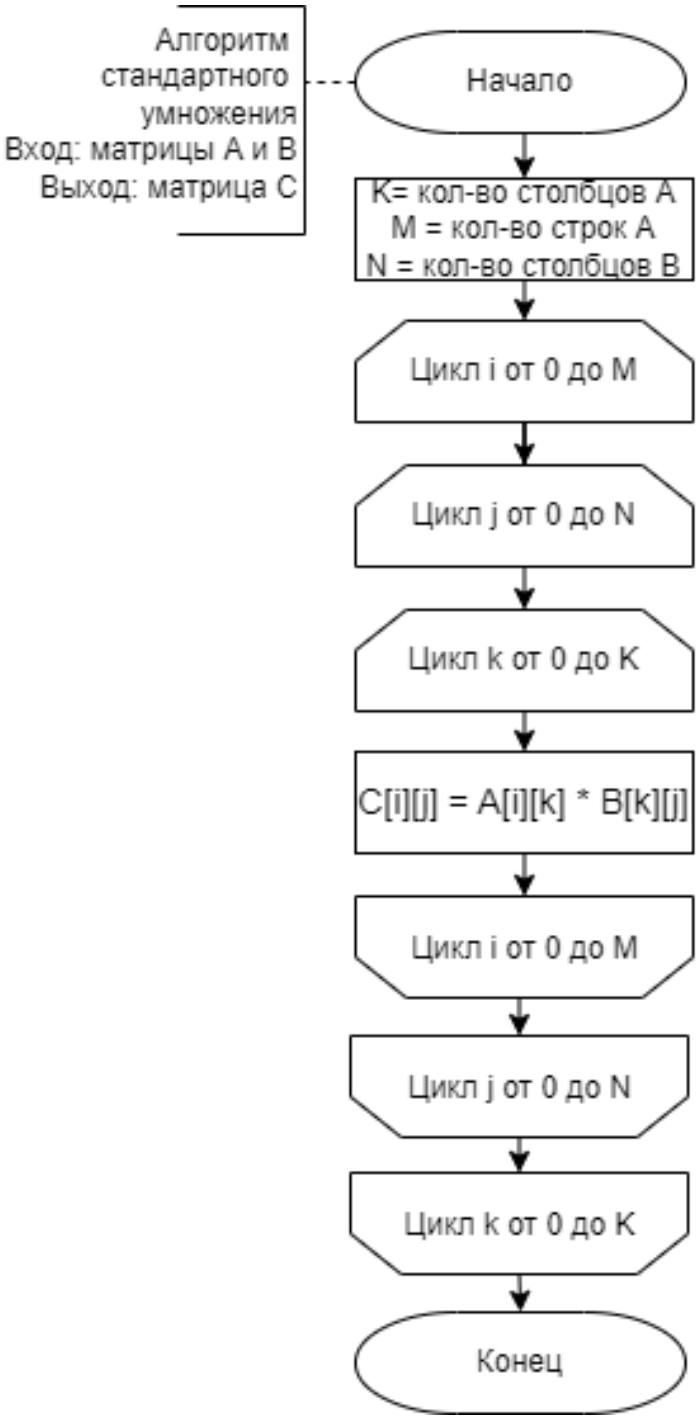

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

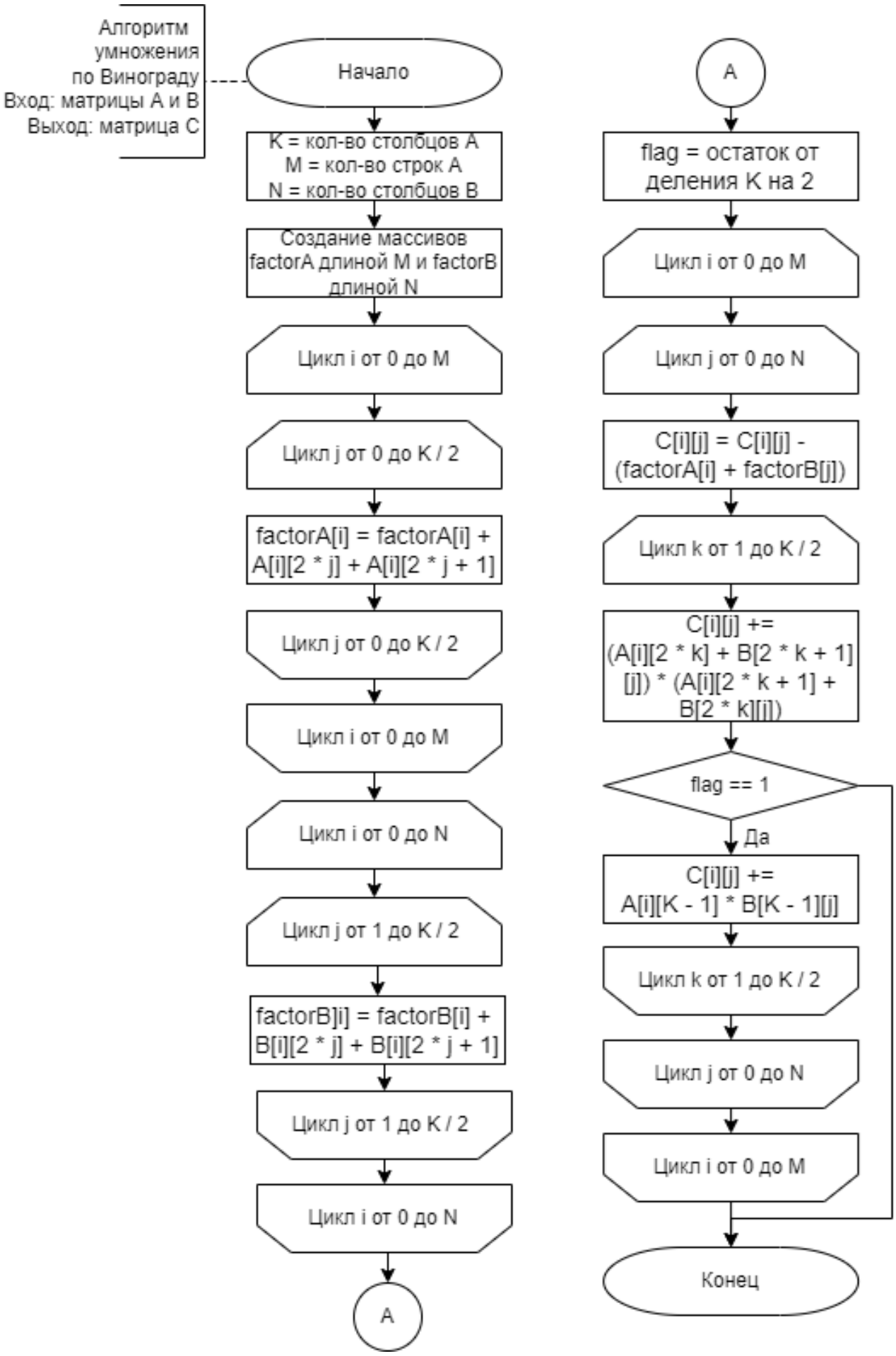

Рисунок 2.2 – Схема алгоритма Винограда

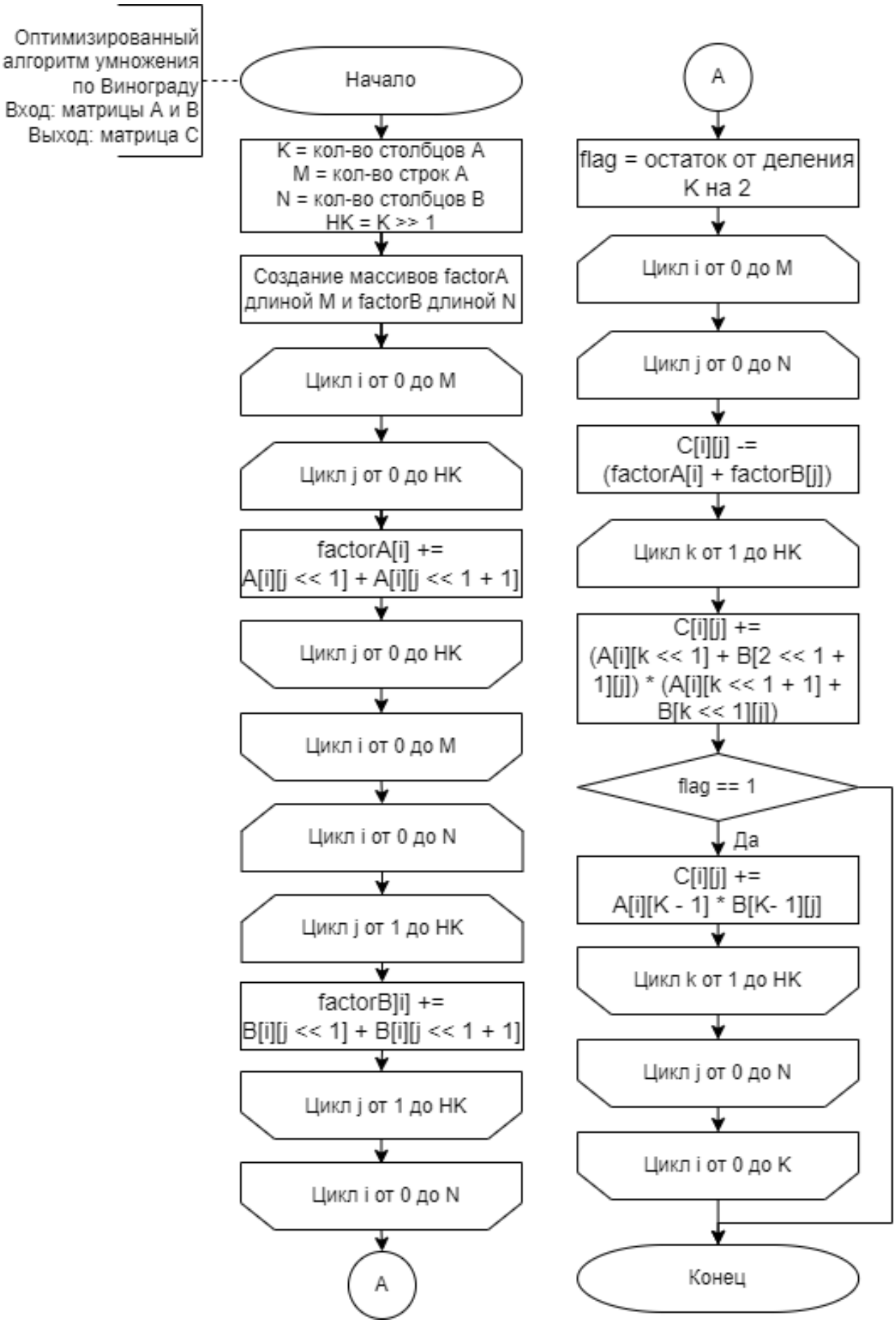

Рисунок 2.3 – Схема оптимизированного алгоритма Винограда

2.2 Модель вычислений трудоемкости

1) Операции, имеющие трудоемкость 1:

$$+, -, \%, ==, != , <, >, <=, >=, [], ++, --, << , >> .$$

2) Операции, имеющие трудоемкость 2:

$$*, /.$$

3) Трудоемкость оператора выбора if условие then A else B рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$

4) Трудоемкость вызова функции равна 0.

5) Трудоемкость цикла рассчитывается как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}).$$

2.3 Трудоёмкость алгоритмов

Пусть M – количество строк в первой матрице, K – количество столбцов в первой матрице, N – количество столбцов во второй матрице.

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из формул, описаных ниже.

1) Цикл по $i \in [0..M)$:

$$f = 2 + M \cdot (2 + f_{body}). \tag{2.1}$$

2) Цикл по $j \in [0..N)$:

$$f = 2 + N \cdot (2 + f_{body}). \tag{2.2}$$

3) Цикл по $k \in [0..K)$:

$$f = 2 + 10K. \tag{2.3}$$

Итого:

$$f_{standard} = 2+M\cdot(4+N\cdot(4+10K)) = 2+4M+4MN+10MNK \approx 10MNK. \tag{2.4}$$

2.3.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из формул, описаных ниже.

1) Создание векторов размерами M и N:

$$f = M + N. \tag{2.5}$$

2) Заполнение вектора M:

$$f = 3 + \frac{K}{2} \cdot (5 + 12M). \tag{2.6}$$

3) Заполнение вектора N:

$$f = 3 + \frac{K}{2} \cdot (5 + 12N). \tag{2.7}$$

4) Цикл заполнения матрицы для чётных размеров:

$$f = 2 + M \cdot (4 + N \cdot (11 + \frac{25}{2} \cdot K)). \tag{2.8}$$

5) Цикл для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f = \begin{cases} 2, & \text{чётная} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.9}$$

Итого:

$$\begin{gathered} f_{wino\_worst} = M + N + 12 + 8M + 5K + \\ +6MK + 6NK + 25MN + \frac{25}{2}MNK \approx 12.5MNK; \end{gathered} \tag{2.10}$$

$$\begin{gathered} f_{wino\_best} = M + N + 10 + 4M + 5K + \\ +6MK + 6NK + 11MN + \frac{25}{2}MNK \approx 12.5MNK. \end{gathered} \tag{2.11}$$

2.3.3 Оптимизированный алгоритм Копперсмита — Винограда

Трудоёмкость оптимизированного алгоритма Копперсмита — Винограда состоит из формул, описаных ниже.

1) Создание вектора M и N:

$$f = M + N. \tag{2.12}$$

2) Заполнения вектора M:

$$f = 2 + \frac{K}{2} \cdot (4 + 8M). \tag{2.13}$$

3) Заполнения вектора N:

$$f = 2 + \frac{K}{2} \cdot (4 + 8N). \tag{2.14}$$

4) Цикл заполнения матрицы для чётных размеров:

$$f = 2 + M \cdot (4 + N \cdot (8 + 9K)). \tag{2.15}$$

5) Цикл для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный.

$$f = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 12N), & \text{иначе.} \end{cases} \tag{2.16}$$

Итого:

$$\begin{gathered} f_{opt\_wino\_worst} = M + N + 10 + 4K + \\ +4MK + 4NK + 8M + 20MN + 9MNK \approx 9NMQ \end{gathered} \tag{2.17}$$

$$\begin{gathered} f_{opt\_wino\_best} = M + N + 8 + 4K + \\ +4MK + 4NK + 4M + 8MN + 9MNK \approx 9NMQ \end{gathered} \tag{2.18}$$

Вывод

В данном разделе были представлены описания стандартного алгоритма умножения матриц, алгоритмом Винограда и оптимизированного алгоритма Винограда, а так же проведена оценка их трудоемкости.

3 Технологическая часть

3.1 Требования к программному обеспечению

В программе должна присутствовать возможность:

1) ввода исходных матриц;

2) умножения матриц стандартным алгоритмом;

3) умножения матриц алгоритмом Винорада;

4) умножения матриц оптимизированным алгоритмом Винорада;

5) замера процессорного времени выполнения реализаций алгоритмов.

3.2 Выбор языка программирования

Для реализации алгоритмов умножения матриц был выбран язык программирования C в силу наличия точных библиотек для замеров процессорного времени и быстродействености языка.

3.3 Выбор библиотеки и способа для замера времени

Для замера процессорного времени выполнения реализаций агоритмов была выбрана не стандартная функция библиотеки <time.h> языка C — clock(), которая недостаточно четко работает при замерах небольших промежутков времени, а QueryPerformanceCounter - API-интерфейс, использующийся для получения меток времени с высоким разрешением или измерения интервалов времени.

Для облегчения работы с данным инструментом были самостоятельно написаны обертки-макросы, представленные на листинге 3.1.

Листинг 3.1 – Листинг макросов

```
1   #define TIMER_INIT \
2       LARGE_INTEGER frequency; \
3       LARGE_INTEGER t1,t2; \
4       double elapsedTime; \
5       QueryPerformanceFrequency(&frequency);
6
7   #define TIMER_START QueryPerformanceCounter(&t1);
8
9   #define TIMER_STOP \
10      QueryPerformanceCounter(&t2); \
11      elapsedTime=(float)(t2.QuadPat1.QuadPart)/frequency.QuadPart/
            COUNT*MICRO; \
12      printf("%lf", elapsedTime);
```

В силу существования явления вытеснения процессов из ядра, квантования процессорного времени все процессорное время не отдается какой-либо одной задаче, поэтому для получения точных результатов необходимо усреднить результаты вычислений: замерить совокупное время выполнения реализации алгоритма N раз и вычислить среднее время выполнения.

3.4 Реализации алгоритмов

В листингах 3.2,3.3,3.4приведены реализации стандартного алгоритма умнлжения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда соответсвенно.

Листинг 3.2 – Листинг стандартного алгоритма умножения матриц

```
matrix_t *mul_matrix_def(matrix_t *mat_1, matrix_t *mat_2)
{
    matrix_t *res = create_matrix(mat_1->rows, mat_2->cols);

    for (int i = 0; i < res->rows; ++i)
        for (int j = 0; j < res->cols; ++j)
            for (int k = 0; k < mat_1->cols; ++k)
                (res->elements)[i][j] +=
                (mat_1->elements)[i][k] * (mat_2->elements)[k][j];

    return res;
}
```

Листинг 3.3 – Листинг алгоритма Винограда

```
matrix_t *mul_matrix_vinograd(matrix_t *mat_1, matrix_t *mat_2)
{
    matrix_t *res = create_matrix(mat_1->rows, mat_2->cols);
    int *row_factor = calloc(mat_1->rows, sizeof(int));
    int *column_factor = calloc(mat_2->cols, sizeof(int));

    for (int i = 0; i < mat_1->rows; ++i)
        for (int j = 0; j < mat_1->cols / 2; ++j)
            row_factor[i] = row_factor[i] + (mat_1->elements)[i][2 * j]*
            (mat_1->elements)[i][2 * j + 1];

    for (int i = 0; i < mat_2->cols; ++i)
        for (int j = 0; j < mat_1->cols / 2; ++j)
            column_factor[i] = column_factor[i] +
            (mat_2->elements)[2 * j][i] * (mat_2->elements)[2 * j+1][i];

    for (int i = 0; i < res->rows; ++i)
        for (int j = 0; j < res->cols; ++j) {
        (res->elements)[i][j] = -row_factor[i] - column_factor[j];
        for (int k = 0; k < mat_1->cols / 2; ++k)
                (res->elements)[i][j] = (res->elements)[i][j] +
                ((mat_1->elements)[i][2 * k] +
                (mat_2->elements)[2 * k + 1][j]) *
                ((mat_1->elements)[i][2 * k + 1] +
                (mat_2->elements)[2 * k][j]);
        }

    if (mat_1->cols % 2 != 0) {
        for (int i = 0; i < res->rows; ++i)
            for (int j = 0; j < res->cols; ++j)
                (res->elements)[i][j] = (res->elements)[i][j] +
                (mat_1->elements)[i][mat_1->cols - 1] *
                (mat_2->elements)[mat_2->rows - 1][j];
        }

    free(row_factor);
    free(column_factor);

    return res;
}
```

Листинг 3.4 – Листинг оптимизированного алгоритма Винограда

```
matrix_t *mul_matrix_vinograd_opt(matrix_t *mat_1, matrix_t *mat_2)
{
    matrix_t *res = create_matrix(mat_1->rows, mat_2->cols);
    int *row_factor = calloc(mat_1->rows, sizeof(int));
    int *column_factor = calloc(mat_2->cols, sizeof(int));

    int eq_dim = mat_1->cols, half_eq_dim = eq_dim >> 1;

    for (int i = 0; i < mat_1->rows; ++i)
        for (int j = 0; j < half_eq_dim; ++j)
            row_factor[i] += (mat_1->elements)[i][j << 1] *
            (mat_1->elements)[i][(j << 1) + 1];

    for (int i = 0; i < mat_2->cols; ++i)
        for (int j = 0; j < half_eq_dim; ++j)
            column_factor[i] += (mat_2->elements)[j << 1][i] *
            (mat_2->elements)[(j << 1) + 1][i];

    for (int i = 0; i < res->rows; ++i)
        for (int j = 0; j < res->cols; ++j) {
            (res->elements)[i][j] = -row_factor[i] - column_factor[j];
            for (int k = 0; k < half_eq_dim; ++k)
                (res->elements)[i][j] += ((mat_1->elements)[i][k << 1]+
                (mat_2->elements)[(k << 1) + 1][j]) *
                ((mat_1->elements)[i][(k << 1) + 1] +
                (mat_2->elements)[(k << 1)][j]);
            }

    if (eq_dim % 2 != 0) {
        for (int i = 0; i < res->rows; ++i)
            for (int j = 0; j < res->cols; ++j)
                (res->elements)[i][j] +=
                (mat_1->elements)[i][mat_1->cols - 1] *
                (mat_2->elements)[mat_2->rows - 1][j];
        }

    free(row_factor); free(column_factor);

    return res;
}
```

3.5 Тестирование алгоритмов

В таблице 3.1 приведены проведенные тесты для функций, реализующих алгоритмы поиска расстояний Левенштейна и Дамерау-Левенштейна.

Таблица 3.1 – Тестирование алгоритмов умножения матриц для разных входный данных

Первая матрица	Вторая матрица	Результирующая матрица
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix}$	$\begin{pmatrix} 14 & 14 \\ 14 & 14 \end{pmatrix}$
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 3 \end{pmatrix}$	$\begin{pmatrix} 6 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{pmatrix}$	$\begin{pmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{pmatrix}$

Вывод

В данном разделе были разработаны стандартный алгоритм умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда, а также проведено тестирование.

4 Экспериментальная часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

1) операционная система Windows-10, 64-bit;
2) оперативная память 8 ГБ;
3) процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz, 2304 МГц, ядер 2, логических процессоров 4.

4.2 Замеры времени

В таблице 4.1 приведены результаты замеров в миллисекундах времени алгоритмов для входных квадратных матриц разной размерности.

Таблица 4.1 – Таблица замера времени выполнения алгоритмов на строках, имеющих разные длины

Размер матрицы	Стандартный	Винограда	Опт. Винограда
50	1.00	0.89	0.75
100	7.94	6.56	5.46
200	65.27	52.25	45.03
300	236.63	190.84	162.55
51	1.06	0.98	0.92
101	8.14	6.60	5.65
201	65.72	52.72	45.48
301	236.30	187.76	168.09

Зависимость времени работы рассматриваемых алгоритмов умножения матриц от размерности матриц (четные размерности) представлена на рисунке 4.1.

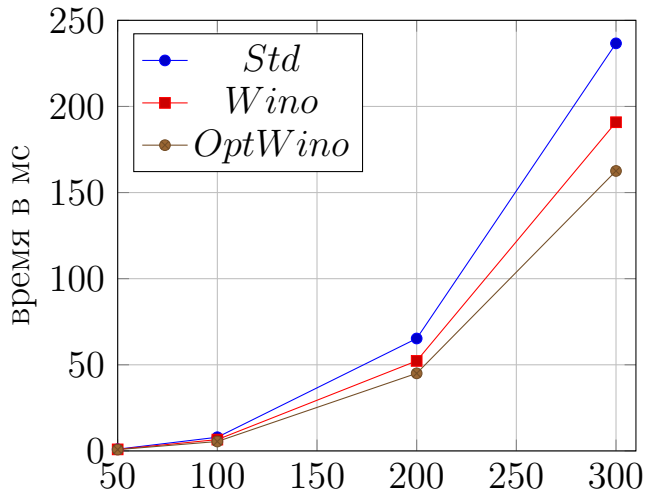

Рисунок 4.1 – Зависимость времени от четной размерности входных матриц

Зависимость времени работы рассматриваемых алгоритмов умножения матриц от размерности матриц (нечетные размерности) представлена на рисунке 4.2.

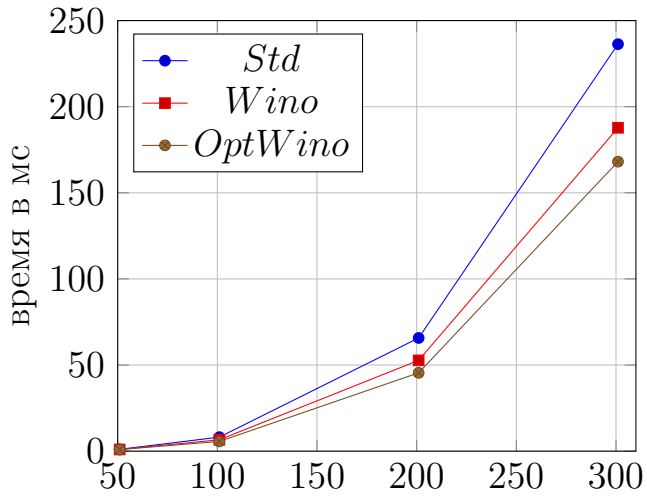

Рисунок 4.2 – Зависимость времени от нечетной размерности входных матриц

Вывод

Результаты замеров показали, что алгоритм Винограда и оптимизированный алгоритм Винограда работают быстрее стандартного алгоритма умножения матриц за счет того, что в них меньше операций умножения. При этом оптимизированный алгоритм Винограда работает немного быстрее обычного алгоритма Винограда за счет введеных оптимизаций.

Заключение

Цель лабораторной работы достигнута – изучены и реализованы алгоритмов умножения матриц (стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда), вычислены их трудоемкости. Все задачи решены:

1) изучены и рассмотрены стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;

2) построены блок-схемы данных алгоритмов;

3) реализован каждый из алгоритмов;

4) оценена трудоемкость алгоритмов;

5) экспериментально оценены временные характеристики алгоритмов;

6) сделан вывод на основании проделанной работы.

На основании проведенных экспериментов и оценки трудоемкости было выявлено, что оптимизированный алгоритм Винограда умножения матриц имеет меньшую сложность чем стандартный алгоритм. И при этом алгоритм Винограда работает быстрее чем стандартный алгоритм в среднем в 1.25 раза, а оптимизированный алгоритм Винограда – в 1.45 раз. Однако для обоих алгоритмов Винограда требуется выделение дополнительной памяти.

Список использованных источников

[1] Умножение матриц. URL: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 09.11.2022).

[2] Алгоритм Копперсмита — Винограда. URL: https://math.fandom.com/ru/wiki/(дата обращения: 10.11.2022).

[3] Microsoft. QueryPerformanceCounter function (profileapi.h). URL: https://learn.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter (дата обращения: 25.10.2022).

[4] The LaTeX project. Core Documentation. URL: https://www.latex-project.org/help/documentation/ (дата обращения: 10.11.2022).