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»

Отчёт

по лабораторной работе №2

Название: Алгоритм Винограда

Дисциплина: Анализ алгоритмов

Студент	ИУ7-52Б		Н. В. Ляпина
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподователь			Л.Л. Волкова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022

Содержание

Введение

Умножение матриц – одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Элементы новой матрицы получаются из элементов старых матриц в соответствии с правилами.

Матрицы A и B могут быть перемножены, если они совместимы в том смысле, что число столбцов матрицы A равно числу строк B. В данной лабораторной работе рассматриваются алгоритмы:

1) классическое умножение матриц;
2) алгоритм Винограда;
3) улучшенный алгоритм Винограда.

Цель лабораторной работы:
реализовать алгоритмы перемножения матриц.

Задачи лабораторной работы:

1) выбрать инструменты для замера процессорного времени выполнения реализации алгоритмов;
2) изучить алгоритмы классического перемножения матриц, Винограда с оптимизацией и без;
3) реализовать:
   а) алгоритм классического перемножения матриц;
   б) алгоритм Винограда;
   в) алгоритм Винограда с оптимизацией.
4) дать оценку трудоёмкости;
5) провести замеры процессорного времени работы и затрачиваемой памяти для всех алгоритмов;
6) привести краткие рекомендации об особенностях применения оптимизаций.

1 Аналитический раздел

В данном разделе будут рассмотрены алгоритмы классического перемножения матриц и Винограда без оптимизации.

1.1 Классическое умножение матриц

Матрицей A размера $[N \cdot M]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая строк N и M столбцов. Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй. [3]

Пусть даны две прямоугольные матрицы A и B размеров $[N \cdot M]$ и $[M \cdot K]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[N \cdot K]$.

$$C_{i,j} = \sum_{l=1}^{M} A_{i,l} \cdot B_{l,j} \tag{1.1}$$

, где i,j – индексы строки и столбца в матрице C соответственно.

1.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.[4]

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $H = (h_1, h_2, h_3, h_4)$.

Скалярное произведение этих векторов равно:

$$V \cdot H = v_1 \cdot h_1 + v_2 \cdot h_2 + v_3 \cdot h_3 + v_4 \cdot h_4 \tag{1.2}$$

Равенство 1.2 можно переписать в виде:

$$V \cdot H = (v_1 + h_2) \cdot (v_2 + h_1) + (v_3 + h_4) \cdot (v_4 + h_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - h_1 \cdot h_2 - h_3 \cdot h_4 \tag{1.3}$$

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. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Вывод

Были рассмотрены алгоритмы классического перемножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и определены способы тестирования.

2.1 Требования к функциональности ПО

В данной работе требуется обеспечить следующую функциональность.

1) Пользовательский режим:
   а) возможность подать на вход две матрицы;
   б) вывод результата умножения трех алгоритмов.
2) Тестовый режим:
   а) возможность проверки корректности работы алгоритмов.
3) Экспериментальный режим:
   а) возможность замера процессорного времени работы алгоритмов для четного и нечетного размеров матриц.

2.2 Схемы алгоритмов

Ниже будут представлены схемы алгоритмов:

1) классическое умножение матриц (рисунок 2.1);
2) алгоритм Винограда (рисунок 2.2);
3) алгоритм Винограда с оптимизацией (рисунок 2.4).

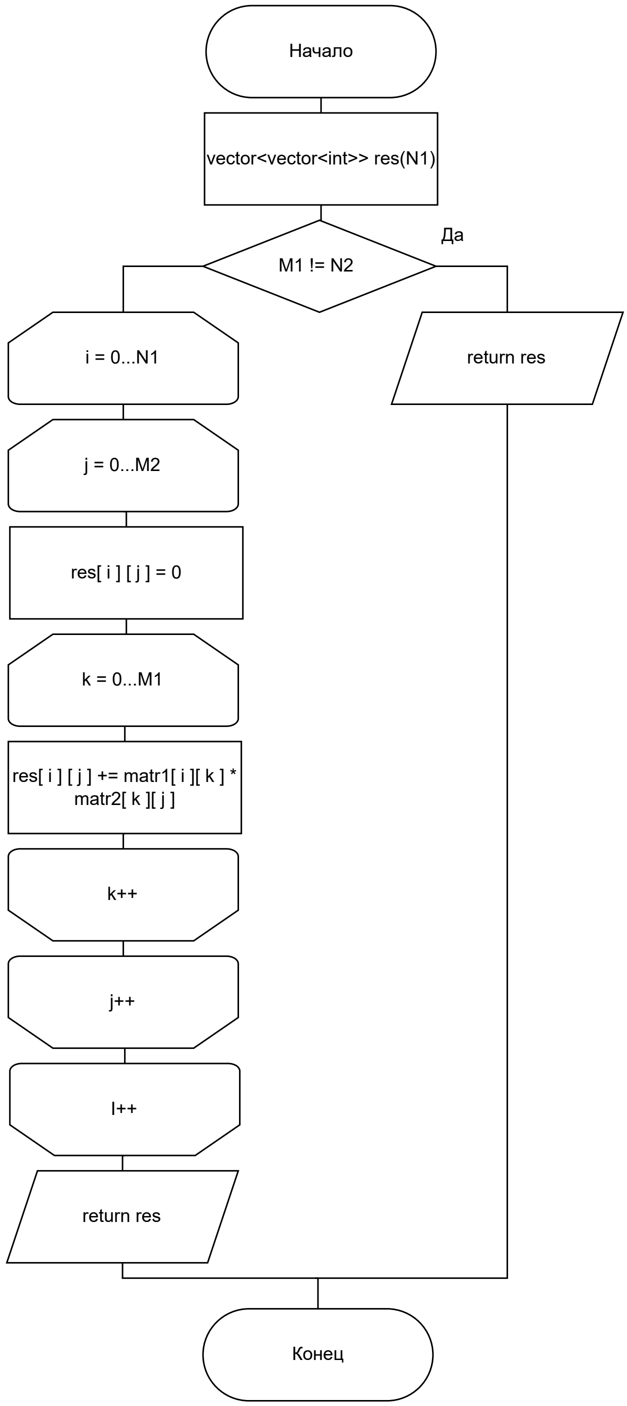

Рисунок 2.1 — Алгоритм классического умножения матриц

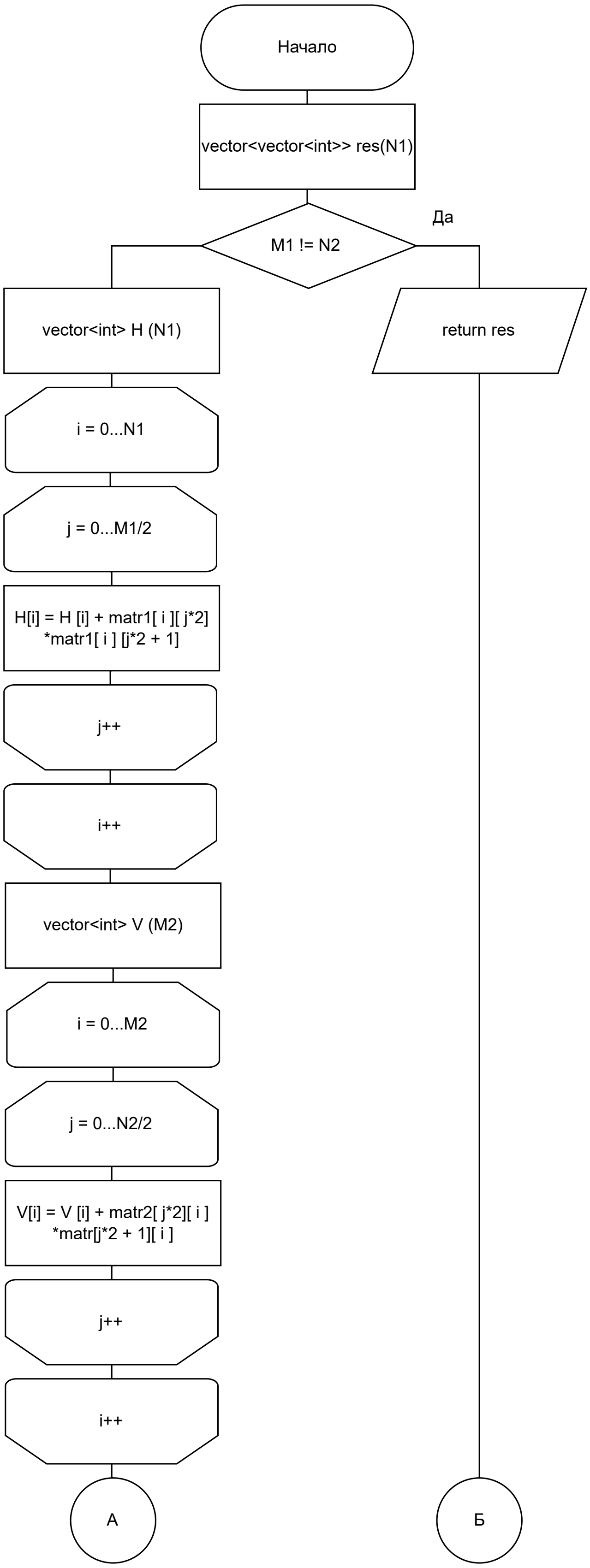

Рисунок 2.2 — Алгоритм Винограда без оптимизации Часть 1

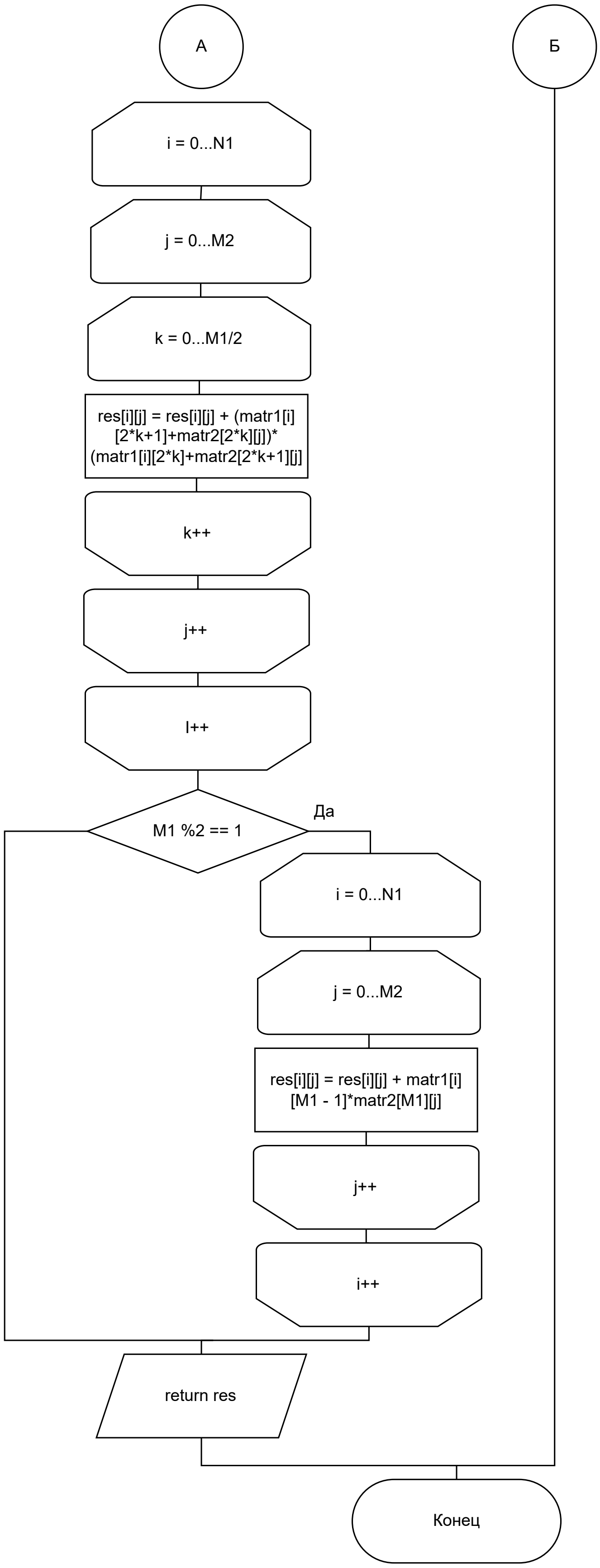

Рисунок 2.3 — Алгоритм Винограда без оптимизации Часть 2

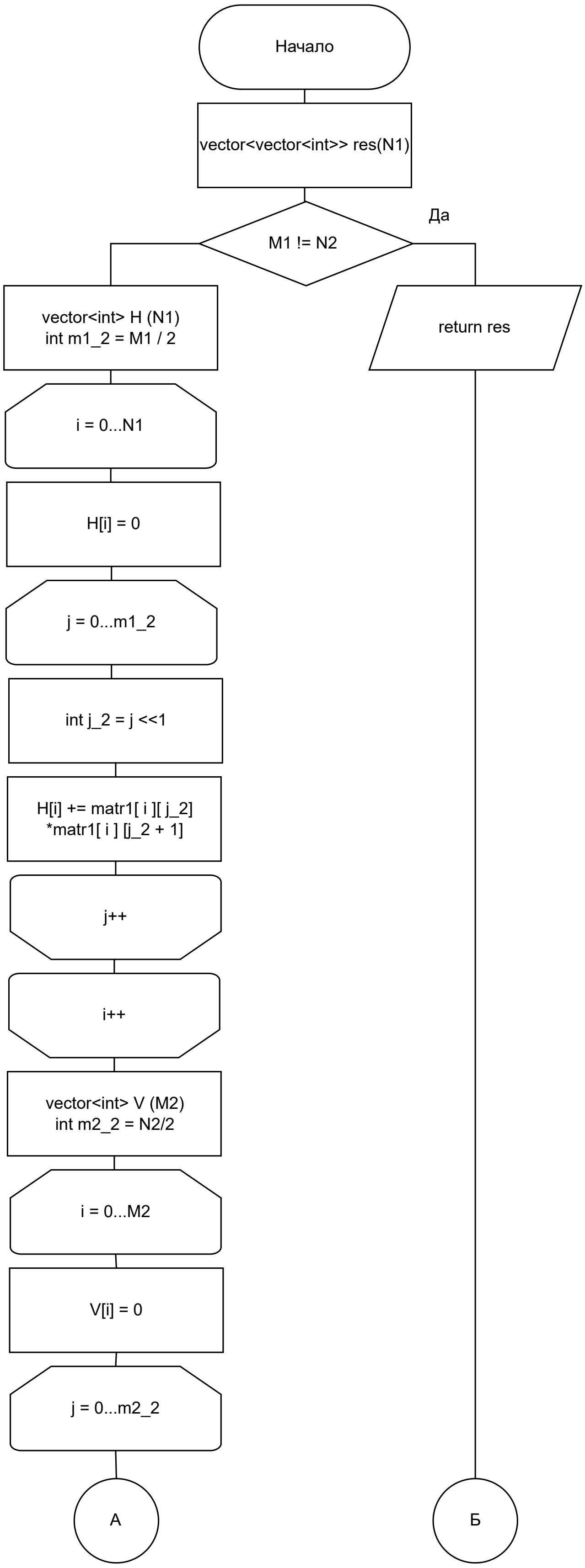

Рисунок 2.4 — Алгоритм Винограда с оптимизацией Часть 1

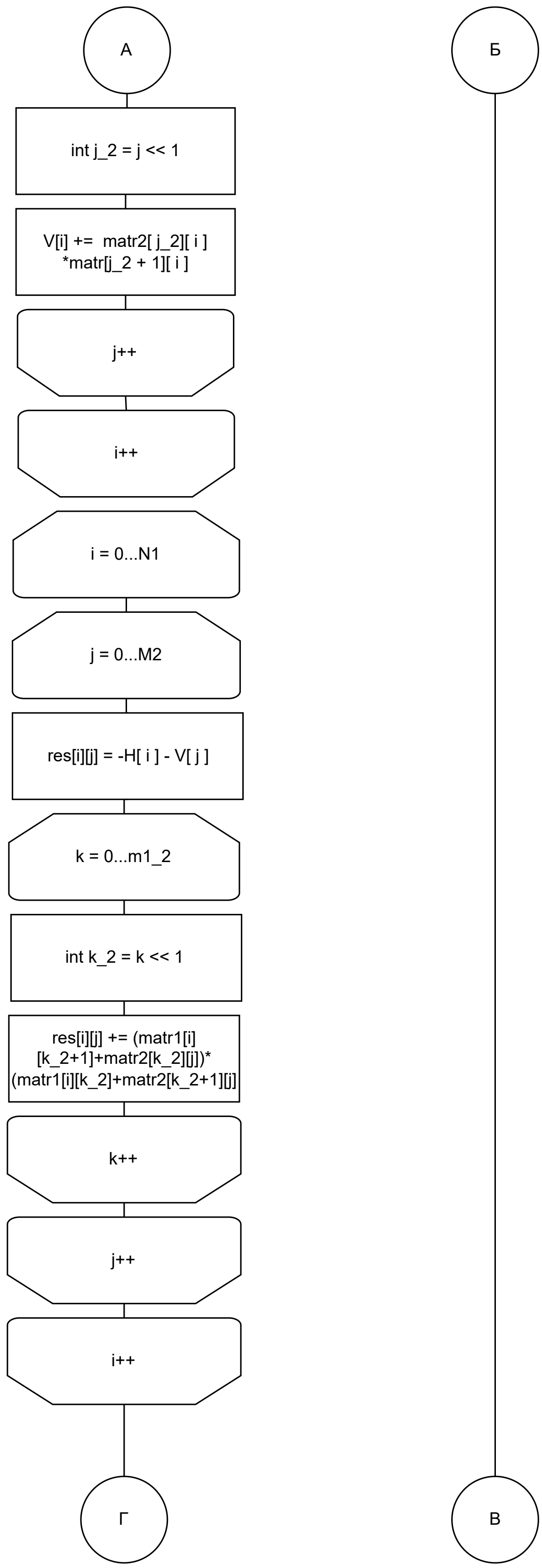

Рисунок 2.5 — Алгоритм Винограда с оптимизацией Часть 2

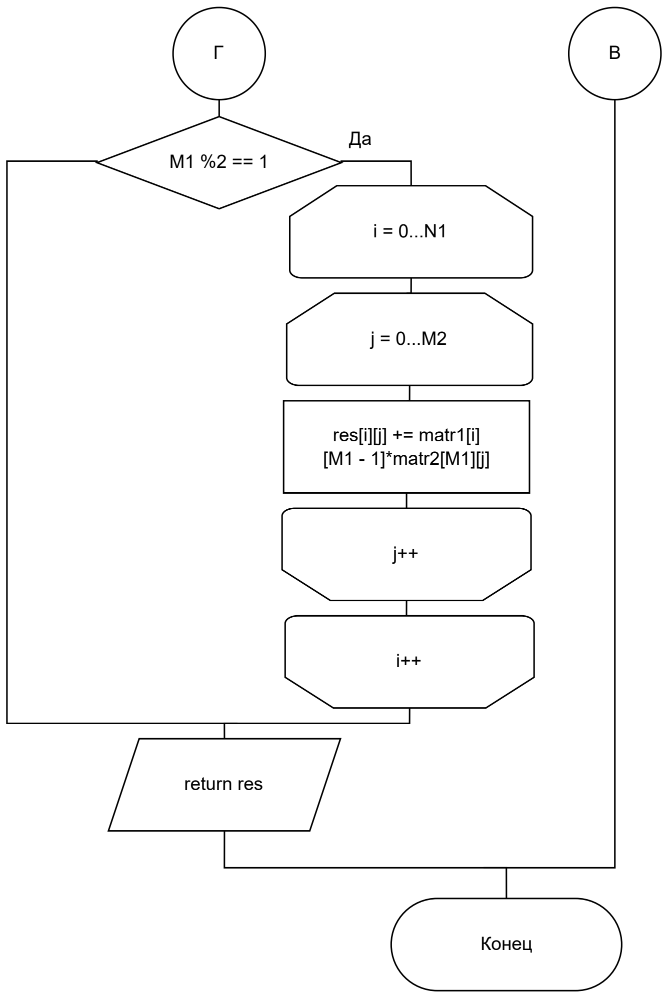

Рисунок 2.6 — Алгоритм Винограда с оптимизацией Часть 3

2.3 Трудоемкость алгоритма

Трудоёмкость – количество работы, которую алгоритм затрачивает на обработку данных. Является функцией от длины входов алгоритма и позволяет оценить количество работы.

Введём модель вычисления трудоёмкости.

2.3.1 Базовые операции

Стоимость представленных ниже операций единична:

1) $=, +, +=, -, -=, *, *=, /, /=, ++, --, \%$
2) $<, \leqslant, >, \geqslant, ==, \neq$
3) $[]$

2.3.2 Условный оператор

$if($ условие $)\{$
// Тело A
$\}$
$else\{$
// Тело B
$\}$

Пусть трудоемкость тела A равна f_A, а тела B f_B, тогда трудоемкость условного оператора можно найти по формуле (2.1):

$$f_{if} = f_{uslovie} + \begin{cases} min(f_A, f_B), & \text{– лучший случай,} \\ max(f_A, f_B), & \text{– худший случай.} \end{cases} \tag{2.1}$$

2.3.3 Цикл со счетчиком

$for(int\ i = 0; i < n; i++)\{$
// Тело цикла
$\}$

Начальная инициализация цикла $int\ i = 0$ выполняется один раз. Условие $i < n$ проверяется перед каждой итерацией цикла и при входе в цикл – $n + 1$ операций. Тело цикла выполняется ровно n раз. Счётчик $i++$ выполняется на каждой итерации, перед проверкой условия, т.е. n раз. Тогда, если трудоёмкость тела цикла равна f, трудоёмкость всего цикла определяется формулой (2.2):

$$f_{for} = 2 + n(2 + f) \tag{2.2}$$

2.3.4 Классический алгоритм

Трудоемкость

$$f_{FirstCycle} = 2 + N1 \cdot (2 + f_{SecondCycle}) \tag{2.3}$$

$$f_{SecondCycle} = 2 + M2 \cdot (2 + f_{ThirdCycle} + 3) \tag{2.4}$$

$$f_{ThirdCycle} = 2 + N1 \cdot (2 + 8) \tag{2.5}$$

$$f_{ClassicMult} = 10 \cdot N1 \cdot M2 \cdot N1 + 7 \cdot N1 \cdot M2 + 4 \cdot N1 + 2 \approx 10 \cdot N1 \cdot M2 \cdot N1 \tag{2.6}$$

2.3.5 Алгоритм Винограда

Трудоемкость

$$f_{FirstCycle} = 2 + N1 \cdot (2 + \frac{M1}{2} \cdot (2 + 12)) = 7 \cdot N1 \cdot M1 + 2 \cdot N1 + 2 \tag{2.7}$$

$$f_{SecondCycle} = 7 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 \tag{2.8}$$

$$f_{ThirdCycle} = 2 + N1 \cdot (2 + M2 \cdot (2 + 7 + \frac{M1}{2} \cdot (2 + 23))) = 12 \cdot N1 \cdot M1 \cdot M2 + 9 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 \tag{2.9}$$

$$f_{if} = 2 + \begin{cases} 0 & \text{– лучший случай,} \\ 15 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 & \text{– худший случай.} \end{cases} \tag{2.10}$$

$$f_{Vinograd} = 12 \cdot N1 \cdot M1 \cdot N1 + 16 \cdot N1 \cdot M2 + 7 \cdot M1 \cdot N1 + 6 \cdot N1 + 6 + \begin{cases} 0 \\ 15 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 \end{cases} \tag{2.11}$$

2.3.6 Оптимизированный алгоритм Винограда

Трудоемкость

$$f_{FirstCycle} = 4 + N1 \cdot (2 + 1 + \frac{M1}{2} \cdot (2 + 10)) = 4 + 3 \cdot N1 + 6 \cdot N1 \cdot M1 \tag{2.12}$$

$$f_{SecondCycle} = 4 + M2 \cdot (2 + 1 + \frac{N2}{2} \cdot (2 + 10)) = 4 + 3 \cdot M2 + 6 \cdot M2 \cdot N2 \tag{2.13}$$

$$f_{ThirdCycle} = 2 + N1 \cdot (2 + M2 \cdot (2 + 7 + \frac{M1}{2} \cdot (2 + 18))) = 2 + 2 \cdot N1 + 9 \cdot N1 \cdot M2 + 10 \cdot N1 \cdot M2 \cdot M1 \tag{2.14}$$

$$f_{if} = 2 + \begin{cases} 0 & \text{– лучший случай,} \\ 13 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 & \text{– худший случай.} \end{cases} \tag{2.15}$$

$$f_{Vinograd} = 5 \cdot N1 + N1 \cdot M2 \cdot (9 + 10 \cdot M1) + 3 \cdot M2 \cdot (1 + 2 \cdot N2) + 6 \cdot N1 \cdot M1 + 10 + \begin{cases} 0 \\ 13 \cdot N1 \cdot M2 + 2 \cdot N1 + 2 \end{cases} \tag{2.16}$$

Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были расчитаны трудоемкости алгоритмов в соответсвии с этой моделью. При схожем коэффициенте при старшем слагаемом в трудоемкости алгоритма Винограда и стандартного, доля долгих операций умножения в алгоритме Винограда меньше.

3 Технологический раздел

В данном разделе представлены средства, использованные в процессе разработки для реализации задачи, а также листинг кода программы. Кроме того показаны результаты тестирования и анализа затрачиваемой памяти.

3.1 Средства реализации

Для реализации поставленной задачи был использован язык C++, так как имеется большой опыт работы с ним. Для измерения процессорного времени была использована ассемблерная вставка.

3.2 Сведения о модулях программы

Программа состоит из:

— main.cpp – главный файл программы, в нем располагается точка входа;
— algs.cpp – содержит алгоритмы умножения матриц;
— time.cpp – содержит ассемблерную вставку для замера времени.

3.3 Листинг программы

В приведенных ниже листингах представлены следующие реализации:

1) классический алгоритм(листинг 3.1);
2) сортировка расчёской (листинг 3.3);
3) сортировка слиянием (листинг 3.5).

Листинг 3.1 — Алгоритм классического умножения

```
vector<vector<int>> ClassicMultMatrix(vector<vector<int>> matr1, vector<vector<int>>
    matr2)
{
    vector<vector<int>> res (matr1.size());

    for(auto &i: res)
        i.resize(matr2[0].size());
    if (matr1[0].size() != matr2.size())
    {
        cout << "Ошибка: кол-во столбцов матрицы 1 != кол-ву строк матрицы 2" <<
            endl;
        return res;
    }
    for (int i = 0; i < matr1.size(); i++)
    {
```

Листинг 3.2 — Алгоритм классического умножения

```
        for (int j = 0; j < matr2[0].size(); j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < matr1[0].size(); k++)
            {
                res[i][j] += matr1[i][k] * matr2[k][j];
            }
        }
    }

    return res;
}
```

Листинг 3.3 — Алгоритм Винограда без оптимизации

```
vector<vector<int>> VinogradMultMatrix(vector<vector<int>> matr1,
    vector<vector<int>> matr2)
{
    vector<vector<int>> res (matr1.size());

    for(auto &i: res)
        i.resize(matr2[0].size());

    if (matr1[0].size() != matr2.size())
    {
        cout << "Ошибка: кол-во столбцов матрицы 1 != кол-ву строк матрицы 2" <<
            endl;
        return res;
    }
    vector<int> H(matr1.size());

    for (int i = 0; i < matr1.size(); i++)
    {
        H[i] = 0;
        for (int j = 0; j < matr1[0].size() / 2; j++)
        {
            H[i] = H[i] + matr1[i][j * 2] * matr1[i][j * 2 + 1];
        }
    }

    vector<int> V (matr2[0].size());

    for (int i = 0; i < matr2[0].size(); i++)
    {
        V[i] = 0;
```

Листинг 3.4 — Алгоритм Винограда без оптимизации

```
        for (int j = 0; j < matr2.size() / 2; j++)
        {
            V[i] = V[i] + matr2[j * 2][i] * matr2[j * 2 + 1][i];
        }
    }
    for (int i = 0; i < matr1.size(); i++)
    {
        for (int j = 0; j < matr2[0].size(); j++)
        {
            res[i][j] = -H[i] - V[j];
            for (int k = 0; k < matr1[0].size() / 2; k++)
            {
                res[i][j] = res[i][j] + (matr1[i][2 * k + 1] + matr2[2 * k][j]) *
                    (matr1[i][2 * k] + matr2[2 * k + 1][j]);
            }
        }
    }

    if (matr1[0].size() % 2 == 1)
    {
        for (int i = 0; i < matr1.size(); i++)
        {
            for (int j = 0; j < matr2[0].size(); j++)
            {
                res[i][j] = res[i][j] + matr1[i][matr1[0].size() - 1] *
                    matr2[matr1[0].size() - 1][j];
            }
        }
    }

    return res;
}
```

Листинг 3.5 — Алгоритм Винограда с оптимизацией

```
vector<vector<int>> VinogradOptimMultMatrix(vector<vector<int>> matr1,
    vector<vector<int>> matr2)
{
    vector<vector<int>> res (matr1.size());

    for(auto &i: res)
        i.resize(matr2[0].size());

    if (matr1[0].size() != matr2.size())
```

Листинг 3.6 — Алгоритм Винограда с оптимизацией

```
    {
        cout << "Ошибка: кол-во столбцов матрицы 1 != кол-ву строк матрицы 2" << endl;
        return res;
    }

    vector<int> H(matr1.size());



    int m1_2 = matr1[0].size()/2;
    for (int i = 0; i < matr1.size(); i++)
    {
        H[i] = 0;
        for (int j = 0; j < m1_2; j++)
        {
            int j_2 = j<<1;
            H[i] += matr1[i][j_2] * matr1[i][j_2 + 1];
        }
    }

    vector<int> V (matr2[0].size());

    int m2_2 = matr2.size()/2;

    for (int i = 0; i < matr2[0].size(); i++)
    {
        V[i] = 0;
        for (int j = 0; j < m2_2; j++)
        {
            int j_2 = j<<1;
            V[i] += matr2[j_2][i] * matr2[j_2 + 1][i];
        }
    }
    for (int i = 0; i < matr1.size(); i++)
    {
        for (int j = 0; j < matr2[0].size(); j++)
        {
            int buf = -H[i] - V[j];
            for (int k = 0; k < matr1[0].size() / 2; k++)
            {
                int k_2 = k << 1;
                buf += (matr1[i][k_2 + 1] + matr2[k_2][j]) * (matr1[i][k_2] + matr2[k_2 + 1][j]);
            }
```

Листинг 3.7 — Алгоритм Винограда с оптимизацией

```
                res[i][j] = buf;
            }
        }

        if (matr1[0].size() % 2 == 1)
        {
            for (int i = 0; i < matr1.size(); i++)
            {
                for (int j = 0; j < matr2[0].size(); j++)
                {
                    res[i][j] += matr1[i][matr1[0].size() - 1] * matr2[matr1[0].size() -
                        1][j];
                }
            }
        }

        return res;
}
```

3.3.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной было реализовано 3 оптимизации:

1) Избавление от умножения на 2 с помощью побитового сдвига;
2) Замена $H[i] = H[i] + ...$ на $H[i] + = ...$

Листинг 3.8 — Оптимизация алгоритма винограда №1 и №2

```
        for (int i = 0; i < matr1.size(); i++)
        {
            H[i] = 0;
            for (int j = 0; j < m1_2; j++)
            {
                int j_2 = j<<1;
                H[i] += matr1[i][j_2] * matr1[i][j_2 + 1];
            }
        }
```

3) Перевычисление некоторых слагаемых алгоритма

Листинг 3.9 — Оптимизация алгоритма винограда №3

```
    for (int i = 0; i < matr1.size(); i++)
    {
        for (int j = 0; j < matr2[0].size(); j++)
        {
            int buf = -H[i] - V[j];
            for (int k = 0; k < matr1[0].size() / 2; k++)
            {
                int k_2 = k << 1;
                buf += (matr1[i][k_2 + 1] + matr2[k_2][j]) * (matr1[i][k_2] + matr2[k_2 + 1][j]);
            }
            res[i][j] = buf;
        }
    }
```

3.4 Тестирование

В таблице 3.1 отображён возможный набор тестов для тестирования методом чёрного ящика, в соответствующих столбцах таблицы представлены результаты тестирования.

Таблица 3.1 — Тесты проверки корректности программы

№	Matrix 1	Matrix 2	Classic	Vinograd	Vinograd Opt
1			NULL	NULL	NULL
2	$\begin{bmatrix} 2 \end{bmatrix}$	$\begin{bmatrix} 2 \end{bmatrix}$	$\begin{bmatrix} 4 \end{bmatrix}$	$\begin{bmatrix} 4 \end{bmatrix}$	$\begin{bmatrix} 4 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 \\ 6 & 6 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 \\ 6 & 6 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 \\ 6 & 6 \end{bmatrix}$
4	$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}$	$\begin{bmatrix} 14 & 14 \\ 14 & 14 \end{bmatrix}$	$\begin{bmatrix} 14 & 14 \\ 14 & 14 \end{bmatrix}$	$\begin{bmatrix} 14 & 14 \\ 14 & 14 \end{bmatrix}$

Вывод

Были разработаны и протестированы спроектированные алгоритмы и указаны средства реализации поставленной задачи.

4 Экспериментальный раздел

В данном разделе будут проведены эксперименты для проведения сравнительного анализа алгоритмов по затрачиваемому процессорному времени в зависимости от размера матрицы.[2]

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта были проведёны следующие эксперименты:

1) сравнение времени работы алгоритмов при нечетном размере массива(график 4.1);
2) сравнение времени работы алгоритмов при четном размере массива(график 4.2).

Тестирование проводилось на компьютере с процессором Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz под управлением Windows 11 с 8 Гб оперативной памяти.

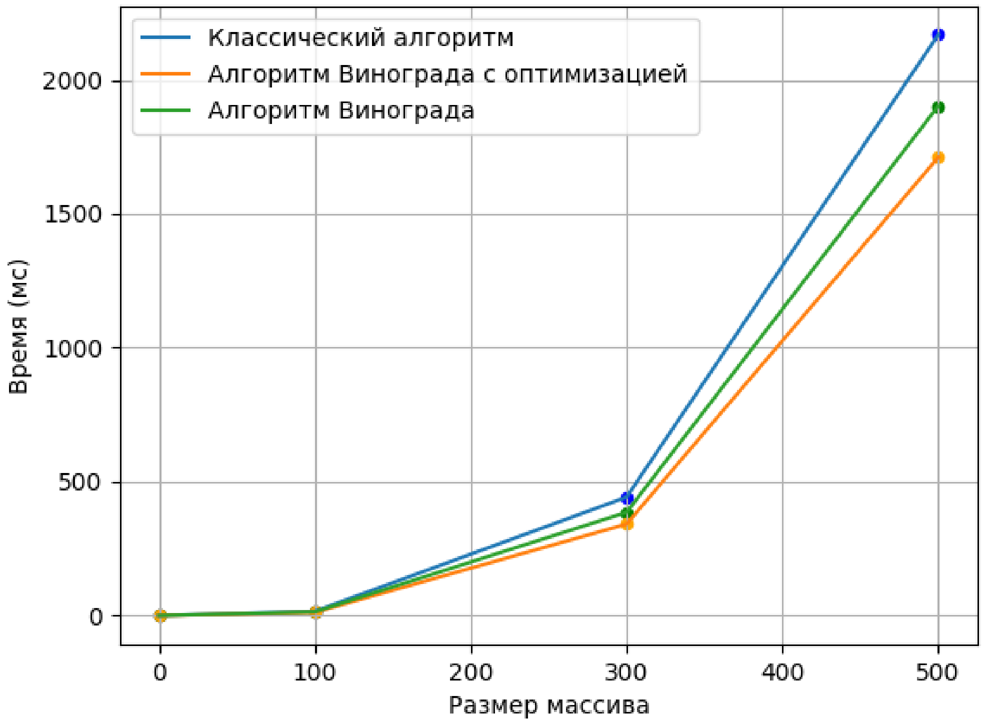

Рисунок 4.1 — Зависимость времени работы алгоритмов от размера матрицы, если он нечетный

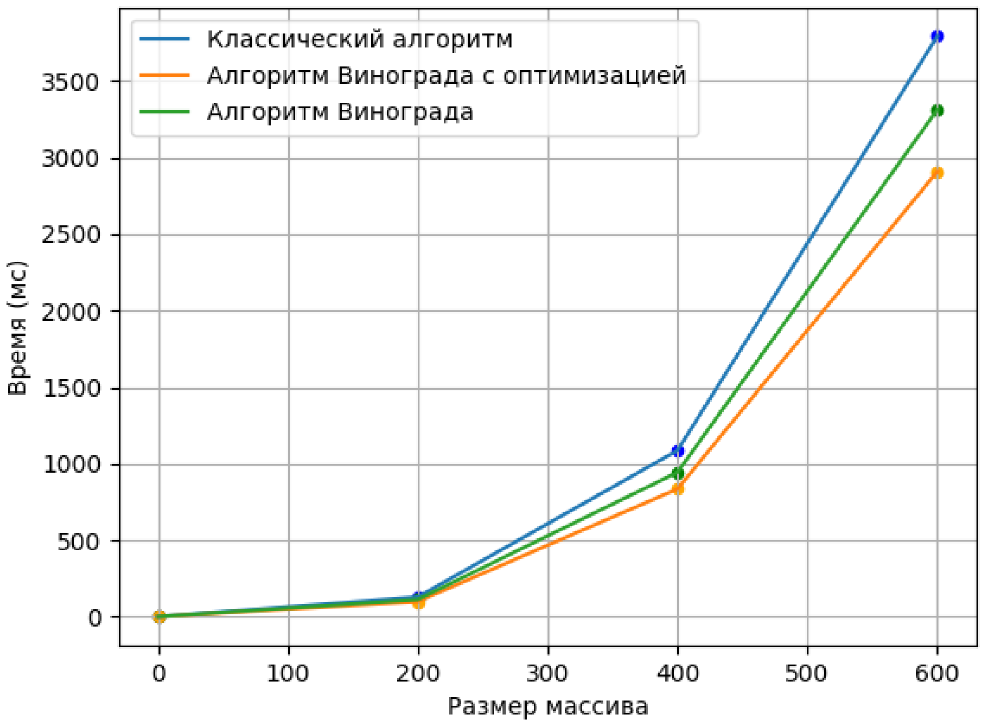

Рисунок 4.2 — Зависимость времени работы алгоритмов от размера матрицы, если он четный

Оптимизация в компиляторе

Аппелируя к графикам выше, можно сделать вывод, что алгоритм с оптимизацией работает быстрее, но важно помнить, что компилятор делает некоторые из этих оптимизаций самостоятельно.

Например, замена умножения на 2 на побитовый влево заменяется компилятором GCC на сложение, а PowerPC делает замену как раз таки на побитовый свдиг.[5]

4.2 Вывод

В ходе экспериментов по замеру времени работы было установлено, что независимо от четного или нечетного размера матрицы алгоритм Винограда с оптимизацией работает быстрее, чем алгоритм Винограда без нее и алгоритм классического умножения матриц.

Заключение

В ходе лабораторной работы были выполнены задачи:

1) В качестве инструмента для замера процессорного времени выполнения алгоритмов была выбрана ассемблерная вставка;
2) Были изучены и реализованы алгоритмы классического перемножения матриц, Виногдара с оптимизацией и без неё;
3) Была дана оценка трудоемкости вышеизложенных алгоритмов;
4) Были проведены замеры процессорного времени работы алгоритмов.

Экспериментально было подтверждено различие во времени работы алгоритмов Винограда и классического алгоритма.

В результате исследования было выяснено, что алгоритм Винограда с оптимизауией может значительно увеличить скорость перемножения матриц, следовательно он более применим в реальных проектах.

Список источников

1) Ассемблерные вставки в AVR-GCC. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/275937/ дата обращения: 20.09.2022);

2) C/C++: как измерять процессорное время. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/282301/ (дата обращения: 20.09.2022);

3) Matrix multiplication. // [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/MatrixMultiplication (дата обращения: 03.10.2022).

4) Умножение матриц по Винограду. // [Электронный ресурс]. Режим доступа: http://www.algolib.narod.ru/Math/Matrix.html (дата обращения: 03.10.2022).

5) Оптимизация в GCC. // [Электронный ресурс]. Режим доступа: https://tproger.ru/translations/will-it-optimize-gcc/ (дата обращения: 03.10.2022).