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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Название: Умножение матриц. Формула Винограда

Дисциплина: Анализ алгоритмов

Студент	ИУ7-54Б		С. Д. Параскун
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Л. Л. Волкова
		Подпись, дата	И. О. Фамилия

Москва, 2021 г.

Содержание

Страница

Введение

В настоящее время компьютеры оперируют множеством типов данных. Одним из них являются матрицы. Для некоторых алгоритмов необходимо выполнять их перемножение. Данная операция является затратной по времени, имея сложность порядка $O(N^3)$. Поэтому Ш. Виноград создал свой алгоритм умножения матриц, который являлся ассимптотически самым быстрым из всех. Однако преимущества появляются только на матрицах большого размера, настолько что современная вычислительная техника не способна оперировать таким объемом данных.

Целью данной лабораторной работы является анализ алгоритмы перемножения матриц Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

- Изучить выбранные алгоритмы умножения матриц и способы оптимизации;
- Составить схемы рассмотренных алгоритмов;
- Реализовать разработанные алгоритмы умножения матриц;
- Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- Описать и обосновать полученные результаты.

1. Аналитический раздел

В данном разделе будут представлены описания алгоритмов умножения матриц [1] стандартным способом, методом Винограда и его оптимизации.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

Индексы у матриц обозначают их размерность - матрица A размером $l \cdot m$, матрица B - $m \cdot n$ соответственно. Перемножать матрицы можно только когда вторая размерность A и первая размерность B совпадают. Тогда результатом умножения данных матриц будет являться матрица C размером $l \cdot n$:

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где каждый элемент матрицы вычисляется следующим образом:

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

1.2 Алгоритм Копперсмита - Винограда

Ассимптотика данного алгоритма является лучшей среди существующих, она составляет $O(N^{2,3755})$ [2].

Рассмотрим два вектора: $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Здесь веткор V играет роль строки первой матрицы-множителя, вектор W - столбца второй матрицы-множителя. Их скалярное произведение равно:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (1.4)$$

Это выражение эквивалентно:

$$V \cdot W = (u_1 + w_2)(u_2 + w_1) + (u_3 + w_4)(u_4 + w_3) - u_1 u_2 - u_3 u_4 - w_1 w_2 - w_3 w_4 \quad (1.5)$$

В случае, если размерность умножаемых векторов нечетна (например, равна 5), добавляется следующее слагаемое:

$$V \cdot W += v_5 w_5 \quad (1.6)$$

Преимуществом данного метода вычисления является то, что слагаемые $v_1 v_2, v_3 v_4, w_1 w_2, w_3 w_4$ можно рассчитать для каждых строки/столбца заранее, тем самым сократить количество операций умножения и привести к менее затратной операции сложения.

Также как варианты оптимизации можно использовать побитовые сдвиги, += , использовать буферы и другое.

1.3 Вывод

В данном разделе были рассмотренны принципы работы алгоритмов умножения матриц. Полученных знаний достаточно для разработки выбранных алгоритмов. На вход алгоритмам будут подаваться две матрицы элементов и их размерность. Реализуемое ПО будет работать в пользо-

вательском режиме (вывод отсортированного массива тремя методами), а также в экспериментальном (проведение замеров времени выполнения алгоритмов).

2. Конструкторский раздел

В данном разделе будут спроектированы схемы алгоритмов, произведена оценка трудоемкости алгоритмов, описаны используемые типы данных, а также произведена оценка памяти и описана структура ПО.

2.1 Схемы алгоритмов

В следующих схемах матрица будет представлена как A, B или Res, а A_{i_j} - j-ый элемент i-ой строки матрицы A.

На рисунках 2.1 - 2.3 представлены схемы рассматриваемых алгоритмов.

В качестве оптимизаций алгоритма Винограда выбрано использование +=, использование переменной-буфера и проверка на нечетность внутри цикла.

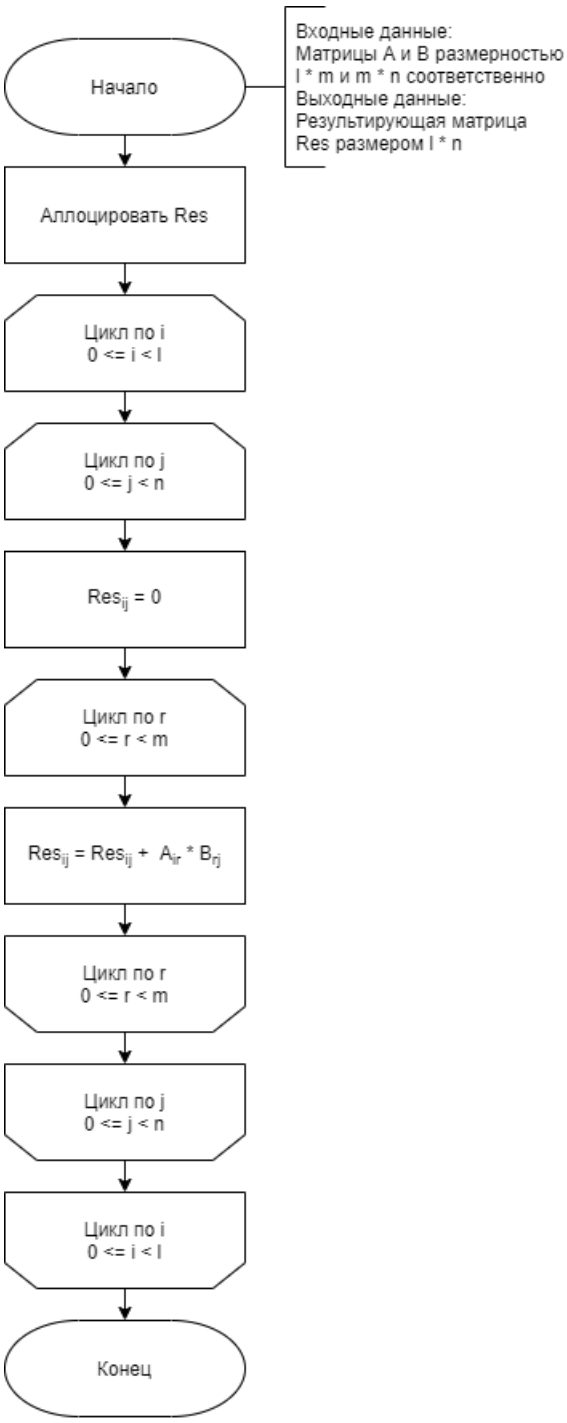

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

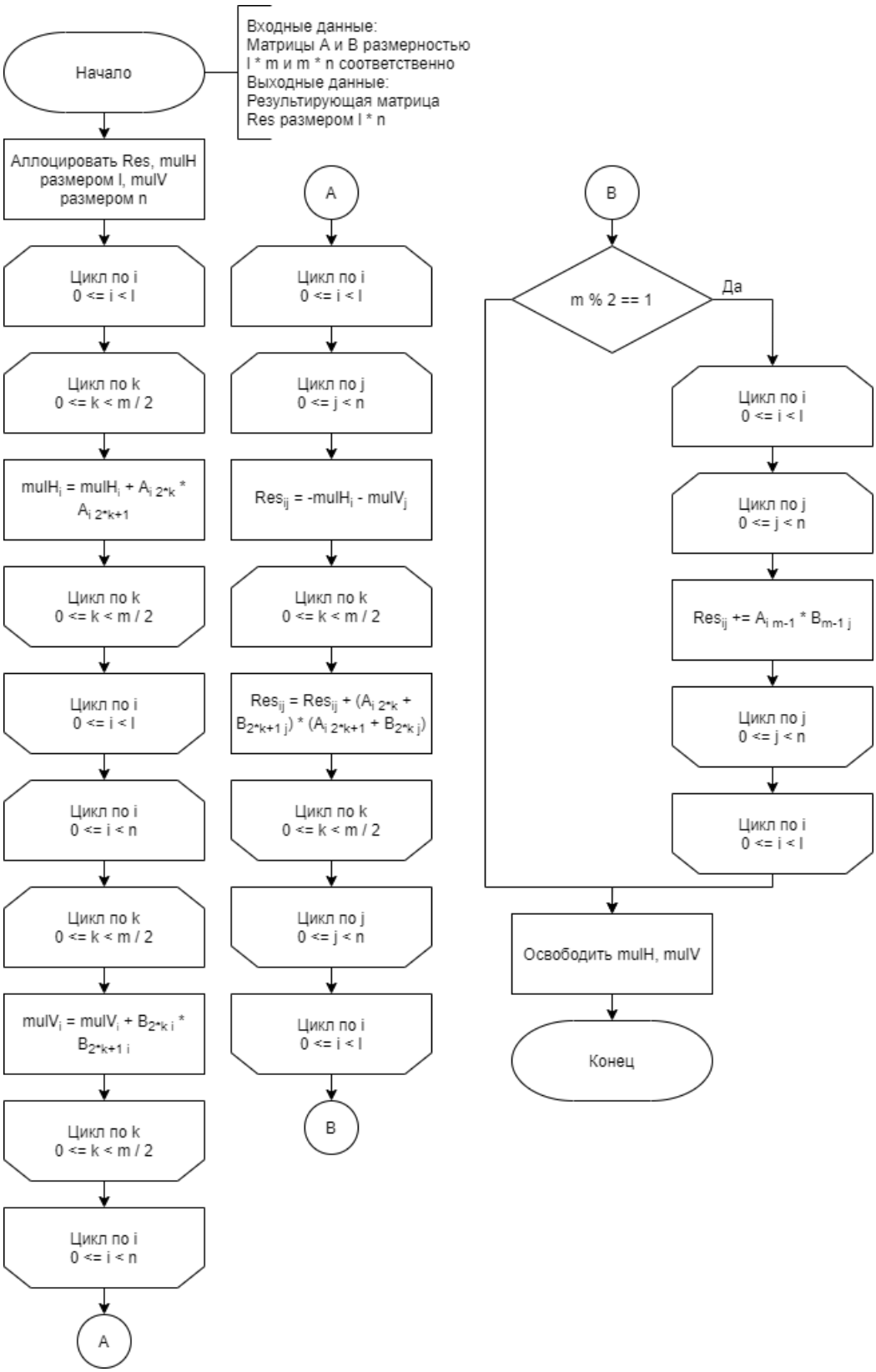

Рисунок 2.2 – Схема алгоритма умножения матриц Винограда

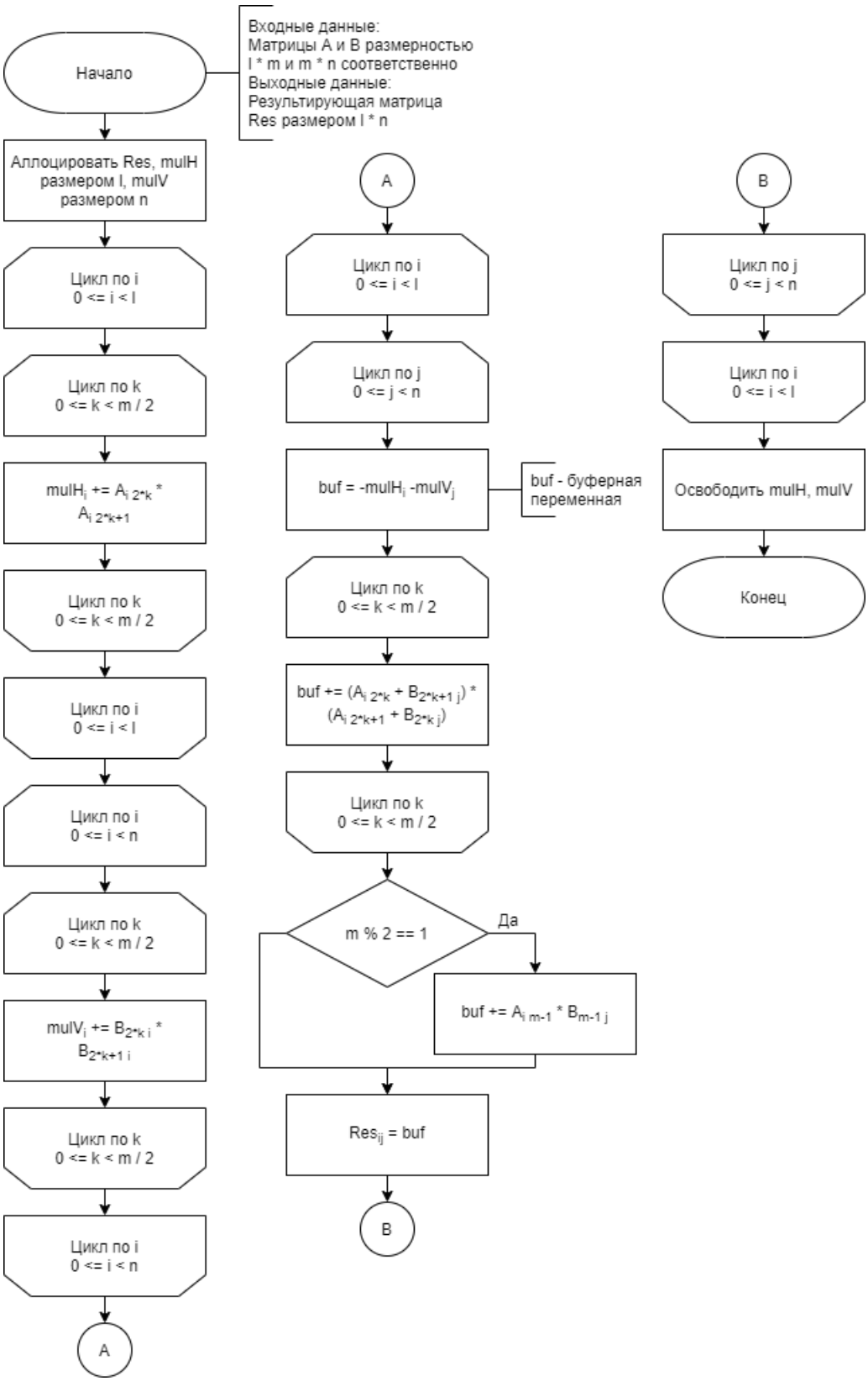

Рисунок 2.3 – Схема алгоритма умножения матриц Винограда (оптимизированный)

2.2 Модель оценки трудоемкости алгоритмов

Трудоемкость алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций
   - Примем трудоемкость следующих операций равной 1:

$$=, +, -, + =, - =, ==, ! =, <, >, \leq, \geq, !, \&\&, ||, [] \tag{2.1}$$

   - Примем трудоемкость следующих операций равной 2:

$$*, /, \%, * =, / =, \tag{2.2}$$

2. Трудоемкость циклов

$$f = f_{init} + f_{compare} + N_{iter} * (f_{body} + f_{inc} + f_{compare}) \tag{2.3}$$

3. Трудоемкость условного оператора Пусть трудоемкость самого условного перехода равна 0. Тогда

$$f_{if} = f_{condition} + \begin{bmatrix} min(f_{true}, f_{false}), \text{в лучшем случае} \\ max(f_{true}, f_{false}), \text{в худшем случае} \end{bmatrix} \tag{2.4}$$

где

f_{init} - трудоемкость инициализации,

$f_{compare}$ - трудоемкость сравнения,

N_{iter} - количество итераций,

f_{body} - трудоемкость тела цикла,

f_{inc} - трудоемкость инкрементации,

$f_{condition}$ - трудоемкость условия,

f_{true}, f_{false} - трудоемкость веток условного оператора.

Трудоемкость станадртного алгоритма

Трудоемкость стандартного алгоритма сосоит из:

- трудоемкость цикла по i от 1 до L: $f = 2 + L(2 + f_{body})$;
- трудоемкость цикла по j от 1 до N: $f = 2 + 3 + N(2 + f_{body})$;
- трудоемкость цикла по r от 1 до M: $f = 2 + M(2 + 12) = 2 + 14M$.

Итого, трудоемкость стандартного алгоритма равна:

$$f = 2+L(2+2+N(2+3+2+14M)) = 2+4L+7LN+14LNM \approx 14LNM \approx O(N^3) \tag{2.5}$$

Трудоемкость алгоритма Винограда

Трудоемкость алгоритма Винограда состоит из:

- аллокация и инициализация векторов mulH и mulV: $f_{HV} = L + N + 2 + L(2 + 3 + \frac{1}{2}M(3 + 14)) + 2 + N(2 + 3 + \frac{1}{2}M(3 + 14)) = 2 + 6L + 6N + \frac{17}{2}LM + \frac{17}{2}NM$;
- цикл по i от 1 до L: $f_L = 2 + L(2 + f_{body})$;
- цикл по j от 1 до N: $f_N = 2 + N(2 + 7 + f_{body})$;
- цикл по k от 1 до M/2: $f_{M/2} = 3 + \frac{1}{2}M(3 + 28) = 3 + \frac{31}{2}M$;
- проверка размеров на нечетность:

$$f_{if} = 3 + \begin{bmatrix} 0, & \text{л.с.} \\ 2 + L(2 + 2 + N(2 + 11)) & \text{х.с.} \end{bmatrix} \tag{2.6}$$

Итого, трудоемкость алгоритма Винограда для лучшего случая равна:

$$f = 4+10L+6N+\frac{17}{2}LM+\frac{17}{2}NM+12LN+\frac{31}{2}LNM \approx 15,5LNM \approx O(N^3) \tag{2.7}$$

Для худшего случая равна:

$$f = 6+14L+6N+\frac{17}{2}LM+\frac{17}{2}NM+25LN+\frac{31}{2}LNM \approx 15,5LNM \approx O(N^3) \tag{2.8}$$

Трудоемкость оптимизированного алгоритма Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

- аллокация и инициализация векторов mulH и mulV: $f_{HV} = L + N + 2 + L(2 + 3 + \frac{1}{2}M(3 + 13)) + 2 + N(2 + 3 + \frac{1}{2}M(3 + 13)) = 2 + 6L + 6N + 8LM + 8NM$;
- цикл по i от 1 до L: $f_L = 2 + L(2 + f_{body})$;
- цикл по j от 1 до N: $f_N = 2 + N(2 + 5 + f_{body} + f_{if} + 3)$;
- цикл по k от 1 до M/2: $f_{M/2} = 3 + \frac{1}{2}M(3 + 23) = 3 + 13M$;
- проверка размеров на нечетность:

$$f_{if} = 3 + \begin{bmatrix} 0, & \text{л.с.} \\ 10 & \text{х.с.} \end{bmatrix} \tag{2.9}$$

Итого, трудоемкость оптимизированного алгоритма Винограда для лучшего случая равна:

$$f = 4 + 10L + 6N + 8LM + 8NM + 16LN + 13LNM \approx 13LNM \approx O(N^3) \tag{2.10}$$

Для худшего случая равна:

$$f = 4 + 10L + 6N + 8LM + 8NM + 26LN + 13LNM \approx 13LNM \approx O(N^3) \tag{2.11}$$

2.3 Используемые типы данных

При реализации алгоритмов будут использованы следующие структуры данных:

- матрица типа int заданного размера;
- размеры матрицы - целое число int;
- вспомогательные массивы типа int заданного размера.

2.4 Оценка памяти

Пусть matrix - матрица целых чисел, l * m и m * n - размерность входных матриц и l * n - результирующей.

Для стандартного алгоритма умножения будет затрачена память на:

- размеры матрицы - 3 * sizeof(int);
- матрицы - (l * m + m * n + l * n) * sizeof(int).

Для алгоритма Винограда создаются два дополнительных массива, для которых используется (l + n) * sizeof(int) памяти. Для оптимазации алгоритма используется вспомогательная переменная buf размером sizeof(int).

Станадартный алгоритм обходится без использования дополнительной памяти, что дает ему выигрыш, хоть и незначительный.

2.5 Структура ПО

ПО будет состоять из следующих модулей:

- Главный модуль - из него будет осуществляться запуск программы и выбор соответствующего режима работы;
- Модуль интерфейса - в нем будет описана реализация режимов работы программы;
- Модуль, содержащий функции работы с данными (такие как генерация данных, вывод, обмен значениями);
- Модуль, содержащий реализации алгоритмов.

2.6 Вывод

На основе полученных в аналитическом разделе знаний об алгоритмах были спроектированы схемы алгоритмов, произведена оценка их трудоемкости, выбраны используемые типы данных, проведена оценка затрачиваемого объема памяти, а также описана структура ПО.

3. Технологический раздел

В данном разделе будут приведены требования к ПО, средства его реализации и листинга кода алгоритмов, а также рассмотрены тестовые случаи.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- ПО принимает две целочисленные матрицы размерностью не больше 700;
- ПО возвращает результирующую матрицу - произведение двух входных матриц.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый язык C. В качестве среды разработки - QtCreator. Оба средства были выбраны из тех соображений, что навыки работы с ними были получены в более ранних курсах.

3.3 Листинги кода

Листинги 3.1 - 3.3 демонстрируют реализацию алгоритмов умножения матриц.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1 int **multStand(int **m1, int **m2, int l, int m, int n)
2 {
3     int **res = allocateMatrix(l, n);
4     if (res)
5     {
6         for (int i = 0; i < l; i++)
7         {
```

```
            for (int j = 0; j < n; j++)
            {
                res[i][j] = 0;
                for (int r = 0; r < m; r++)
                    res[i][j] = res[i][j] m1[i][r] * m2[r][j];
            }
        }
    }
    else
        printf("Error!\n");
    return res;
}
```

Листинг 3.2 – Алгоритм умножения матриц Винограда

```
int **multVin(int **m1, int **m2, int l, int m, int n)
{
    int **res = allocateMatrix(l, n);
    int *mulH = calloc(l, sizeof(int));
    int *mulV = calloc(n, sizeof(int));
    if (res && mulH && mulV)
    {
        for (int i = 0; i < l; i++)
            for (int k = 0; k < m / 2; k++)
                mulH[i] = mulH[i] + m1[i][2 * k] * m1[i][2 * k + 1];

        for (int i = 0; i < n; i++)
            for (int k = 0; k < m / 2; k++)
                mulV[i] = mulV[i] + m2[2 * k][i] * m2[2 * k + 1][i];

        for (int i = 0; i < l; i++)
            for (int j = 0; j < n; j++)
            {
                res[i][j] = -mulH[i] - mulV[j];
                for (int k = 0; k < m / 2; k++)
                    res[i][j] = res[i][j] + (m1[i][2 * k]\
                            + m2[2 * k + 1][j]) *\
                            (m1[i][2 * k + 1] + m2[2 * k][j]);
            }

        if (m % 2 == 1)
            for (int i = 0; i < l; i++)
                for (int j = 0; j < n; j++)
                    res[i][j] += m1[i][m - 1] * m2[m - 1][j];

        free(mulH);
        free(mulV);
    }
```

```
    else
        printf("Error!\n");
    return res;
}
```

Листинг 3.3 – Алгоритм умножения матриц Винограда (оптимизированный)

```
int **multVinOptimize(int **m1, int **m2, int l, int m, int n)
{
    int **res = allocateMatrix(l, n);
    int *mulH = calloc(l, sizeof(int));
    int *mulV = calloc(n, sizeof(int));
    int buf;
    if (res && mulH && mulV)
    {
        for (int i = 0; i < l; i++)
            for (int k = 0; k < m / 2; k++)
                mulH[i] += m1[i][2 * k] * m1[i][2 * k + 1];

        for (int i = 0; i < n; i++)
            for (int k = 0; k < m / 2; k++)
                mulV[i] += m2[2 * k][i] * m2[2 * k + 1][i];

        for (int i = 0; i < l; i++)
            for (int j = 0; j < n; j++)
            {
                buf = 0;
                buf = -mulH[i] - mulV[j];
                for (int k = 0; k < m / 2; k++)
                    buf += (m1[i][2 * k] + m2[2 * k + 1][j])\
                            * (m1[i][2 * k + 1] + m2[2 * k][j]);
                if (m % 2 == 1)
                    buf += m1[i][m - 1] * m2[m - 1][j];
                res[i][j] = buf;
            }

        free(mulH);
        free(mulV);
    }
    else
        printf("Error!\n");
    return res;
}
```

3.4 Тестирование ПО

В таблице 3.1 приведены тестовые случаи для алгоритмов умножения матриц. Случаи 1 - 2 - умножение однострочных/одностолбцовых матриц, 3 - 5 - матрицы одного размера, 6 - некорректный размер.

Таблица 3.1 – Тестовые случаи

N	Матрица 1	Матрица 2	Результат
1	$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 5 & 5 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 15 \\ 6 \end{pmatrix}$
2	$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$
4	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
5	$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
6	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	Некорректный размер

3.5 Вывод

На основе схем из конструкторского раздела были написаны реализации требуемых алгоритмов, а также проведено их тестирвание.

4. Исследовательский раздел

В данном разделе будет проведен сравнительный анализ алгоритмов по времени и затрачиваемой памяти.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими характеристиками:

- операционная система Windows 10 [3]
- память 8 Гб
- процессор Intel Core i3 7020U, 2.3 ГГц [4]

Во время проведения эксперимента устройство не было нагружено сторонними задачами, а также было подключено к блоку питания.

Замеры процессорного времени проводились с помощью ассемблерной вставки, вычисляющей затраченное процессорное время в тиках.

Листинг 4.1 – Ассемблерная вставка замера процессорного времени в тиках

```
unsigned long long tick(void)
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc": "=A" (d));
    return d;
}
```

При проведении эксперимента была отключена оптимизация командой компилятору -o0 [5].

4.2 Оценка времени работы алгоритмов

В таблицах 4.1 - 4.3 представлены замеры процессорного времени работы алгоритмов на квадратных матрицах размером от 50 до 550 элементов. Та-

кая размерность выбрана, потому что на больших размерах массива происходит переполнение типа счетчика тиков. Каждое значение было получено усреднением по 10 замерам.

Примем за лучший случай четную размерность, за худший - нечетную.

Таблица 4.1 – Временные замеры работы алгоритмов для лучшего случая

N	Standard	Vinograd	Vinograd (optimize)
50	1000000	1000000	1000000
100	15000000	12000000	10000000
150	63000000	52000000	40000000
200	185000000	152000000	120000000
250	418000000	341000000	270000000
300	844000000	693000000	545000000
350	1572000000	1297000000	1023000000
400	2630000000	2168000000	1709000000
450	4187000000	3449000000	2723000000
500	6434000000	5462000000	4481000000
550	9540000000	7948000000	6432000000

Таблица 4.2 – Временные замеры работы алгоритмов для худшего случая

N	Standard	Vinograd	Vinograd (optimize)
51	1000000	1000000	1000000
101	18000000	12000000	9000000
151	66000000	51000000	40000000
201	186000000	149000000	117000000
251	443000000	358000000	283000000
301	864000000	707000000	557000000
351	1563000000	1282000000	1011000000
401	2678000000	2188000000	1717000000
451	4491000000	3719000000	2912000000
501	6669000000	5498000000	4318000000
551	9705000000	7972000000	6261000000

На рисунке 4.1 представлен график сравнения времени работы алгоритмов на случайных данных. В результате эксперимента было получено, что на квадратных матрицах размером до 550 оптимизированный алгоритм Винограда работает на 40% быстрее стандартного и на 25% быстрее классического алгоритма Винограда.

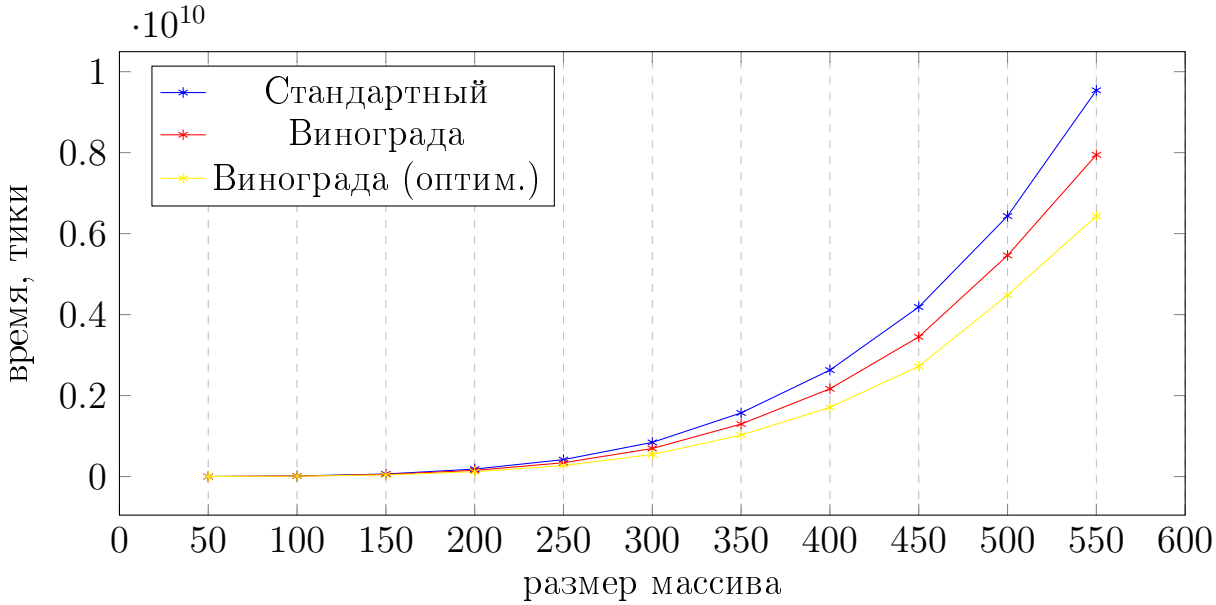

Рисунок 4.1 – Сравнение времени работы алгоритмов для лучшего случая

На рисунке 4.2 представлен график сравнения времени работы алгоритмов на отсортированных данных. В результате эксперимента было получено, что на квадратных матрицах размером до 551 алгоритмы показзывают примерно такие же результаты.

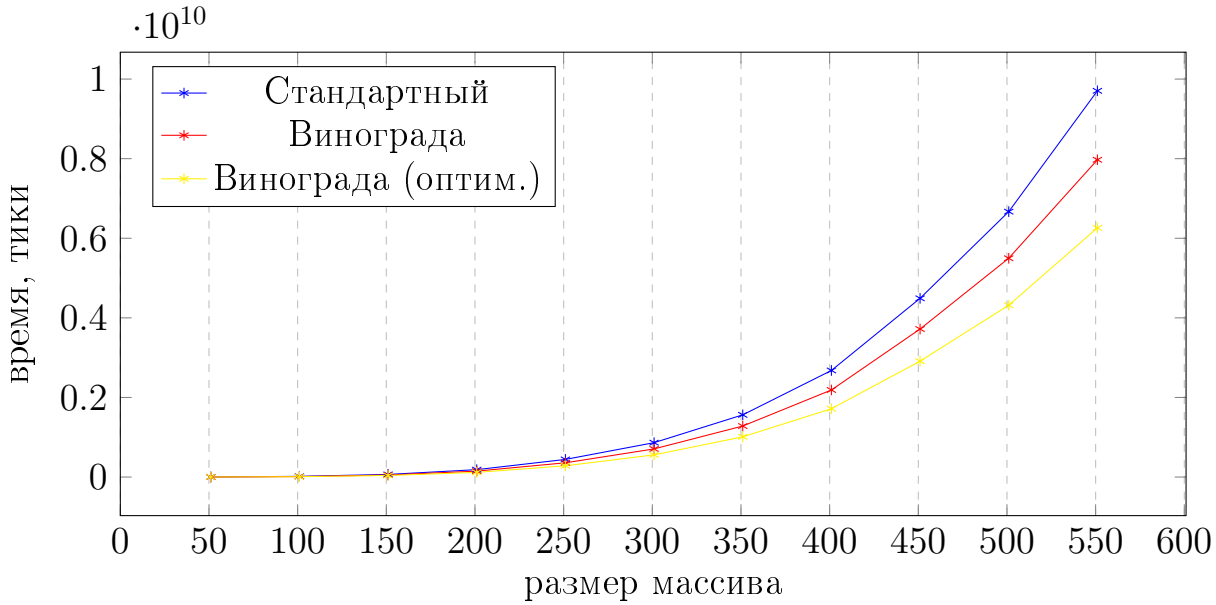

Рисунок 4.2 – Сравнение времени работы алгоритмов для худшего случая

4.3 Вывод

Реализиция стандартного алгоритма умножения матриц:

- в лучшем случае - на 15% медленнее алгоритма Винограда и на 30% медленнее оптимизированного алгоритма Винограда;
- в худшем случае - на 18% медленнее алгоритма Винограда и на 35% медленнее оптимизированного алгоритма Винограда.

Также оптимизированный алгоритм Винограда дает выигрыш на 20% относительно стандартного алгоритма Винограда независимо от данных.

Заключение

В ходе выполнения данной лабораторной работы были выполнены следующие задачи:

- Изучены выбранные алгоритмы умножения матриц и способы оптимизации;
- Составлены схемы рассмотренных алгоритмов;
- Реализованны разработанные алгоритмы умножения матриц;
- Проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- Описаны и обоснованы полученные результаты.

В результате исследований можно прийти к выводу, что стандартный алгоритм является самым медленным из рассмотренных. Хотя сложность всех алгоритмов равняется $O(N^3)$, оптимизированный алгоритм Винограда работает быстрее в среднем на 25% при любых данных. По используемой памяти алгоритмы практически не отличаются, за исключением использования двух дополнительных массивов в алгоритме Винограда.

Список литературы

[1] Матрица [Электронный ресурс]. Режим доступа: https://terme.ru/termin/matrica.html (дата обращения: 27.10.2021).

[2] Умножение матриц [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 27.10.2021).

[3] Windows 10 [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/windows/windows-10-specifications (дата обращения: 18.10.2021).

[4] Процессор Intel Core i3-7020u [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/122590/intel-core-i37020u-processor-3m-cache-2-30-ghz/specifications.html (дата обращения: 18.10.2021).

[5] ISO/IEC 9899:1999 [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf (дата обращения: 22.10.2021).