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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

по лабораторной работе № 2

Название: Анализ алгоритмов умножения матриц

Дисциплина: Анализ алгоритмов

Студент	ИУ7-52Б		Короткая В. М.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Волкова Л.Л.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

Содержание

Введение

В этой лабороторной работе мы рассматриваем вопрос, который часто **Умножение матриц** - это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверточных слоях неронной сети базируются на этой операции. Для перемножения двух матриц необходимо, чтобы количество столбцов в первой матрице совпадало с количеством строк во второй. У результирующей матрицы будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Сложность вычисления произведения матриц по определению составляет $O(n^3)$, однако существуют более эффективные алгоритмы, которые применяются для больших матриц. Вопрос о предельной скорости умножения больших матриц, также как и вопрос о построении наиболее быстрых и устойчивых практических алгоритмов умножения больших матриц остаётся одной из нерешённых проблем линейной алгебры.

Цель данной лабораторной работы заключается в изучении алгоритмов умножения матриц. Рассматриваются стандартный алгоритм умножения матриц, а также алгоритм Винограда и модифицированный алгоритм Винограда. Требуется рассчитать и изучить затрачиваемое каждым алгоритмом время.

В данной лабораторной работе выделено несколько задач:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- модифицировать алгоритм Винограда;
- дать теоретическую оценку базового алгоритма умножения матриц, алгоритму Винограда и модифированному алгоритму Винограда;
- реализовать три алгоритма умножения матриц на одном из языков программирования;
- сравнить алгоритмы умножения матриц.

1. Аналитическая часть

В данном разделе будут рассмортенно формальное описание алгоритмов.

1.1. Стандартный алгоритм умножения матриц

Пусть даны две матрицы A и B с размерностями $m \times n$ и $n \times l$ соответственно (1.1) и (1.2):

$$\begin{bmatrix} a_{1,1} & ... & a_{1,n} \\ ... & ... & ... \\ a_{m,1} & ... & a_{m,n} \end{bmatrix} \tag{1.1}$$

$$\begin{bmatrix} b_{1,1} & ... & b_{1,l} \\ ... & ... & ... \\ b_{n,1} & ... & b_{n,l} \end{bmatrix} \tag{1.2}$$

В результате умножения, получим матрицу C размерностью $m \times l$ (1.3):

$$\begin{bmatrix} c_{1,1} & ... & c_{1,l} \\ ... & ... & ... \\ c_{m,1} & ... & c_{m,l} \end{bmatrix} \tag{1.3}$$

$c_{i,j} = \sum\limits_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B.

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.

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 (1.4).

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.4}$$

Это равенство можно переписать в виде ()1.5)

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (1.5)$$

Кажется, что формула 1.5 задает больше работы, чем первое: вместо четырех умножений мы насчитываем их шесть, а вместо трех сложений - десять. Менее очевидно, что 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. На практике это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Вывод

В данном раздели были рассмотрены алгоритмы умножения матриц. Алгоритм Винограда предлагает подсчитывать значения заранее перед основными вычислениями, что может повысить производительность при перемножении достаточно больших матриц. Однако с малыми размерами, он может справляться хуже чем другие, но алгоритм можно оптимизировать и добиться более высокой скорости подсчёта.

Входными данными реализуемого ПО являются:

- размерность первой матрицы - два натуральных числа;
- первая матрица - целочисленная;
- размерность второй матрицы - два натуральных числа;
- вторая матрица - целочисленная.

Выходными данными реализуемого ПО являеться результат алгоритмов умножения матриц т. е.:

- матрица (результат) - для классического умножения матриц;
- матрица (результат) - для алгоритма Винограда;

Ограничением для реализуемого ПО является - размерность вводимых матриц т. е. длинна строки первой матрицы должна совпадать с длинной колонки второй матрицы.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут описаны пользовательские структуры данных, приведены структура ПО и классы эквивалентности для тестирования реализуемого ПО.

2.1. Схемы алгоритмов

На рисунке 2.1 представлена схема классического алгоритма умножения матриц.

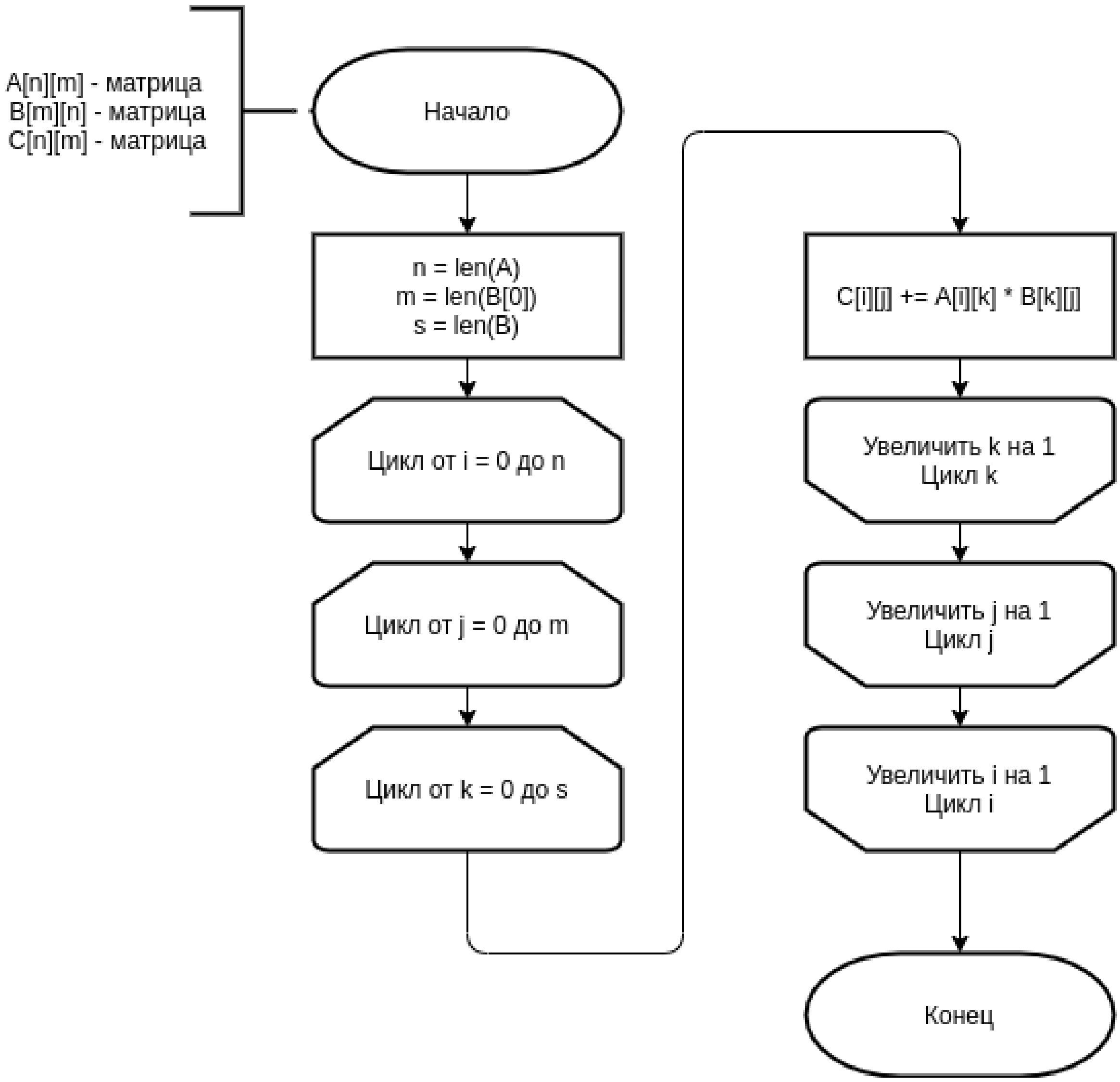

Рис. 2.1: Классический алгоритм умножения матриц.

,

На рисунках 2.2-2.3 представлена схема алгоритма Винограда умножения матриц.

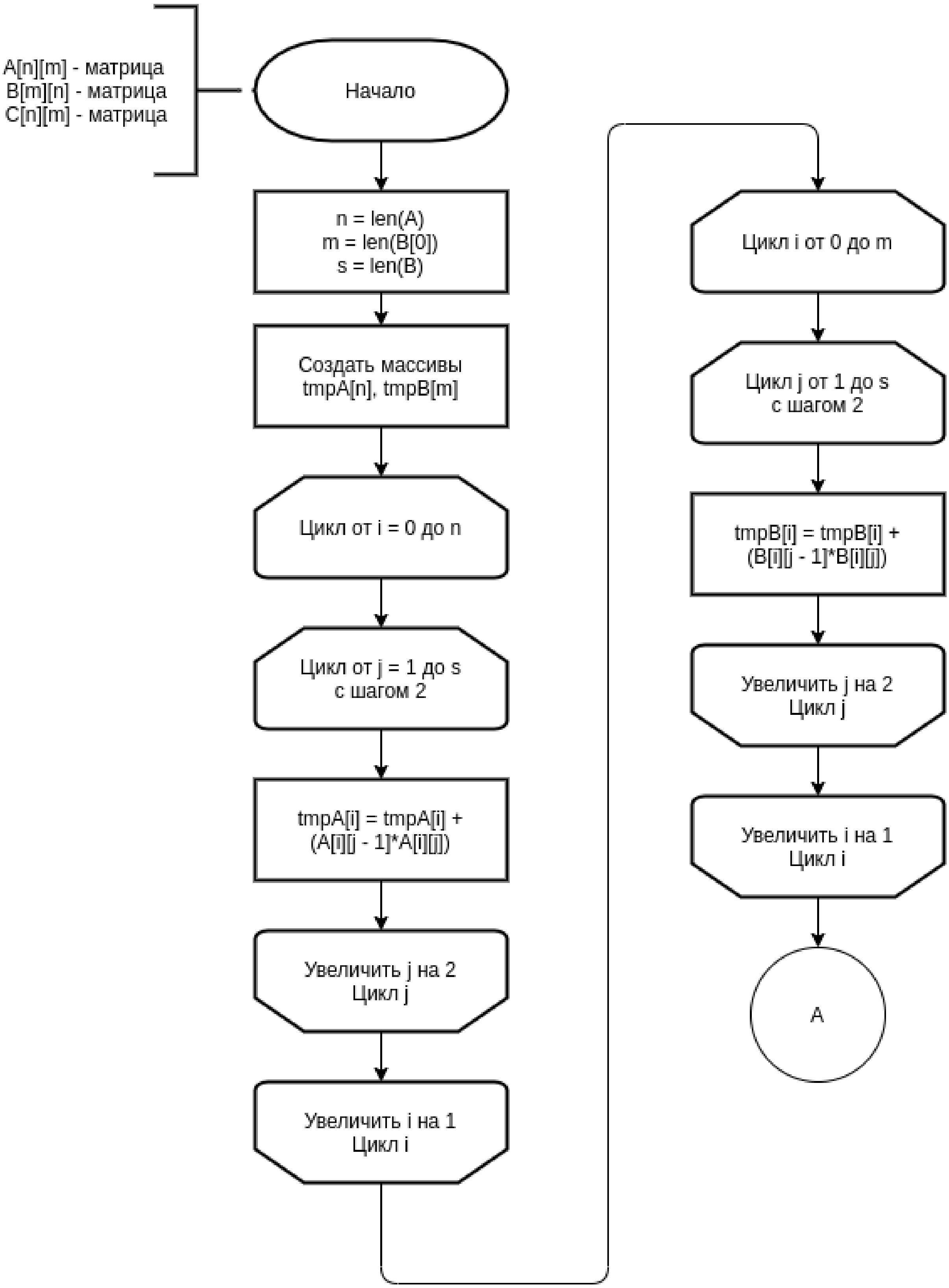

Рис. 2.2: Алгоритм Винограда умножения матриц(часть 1)

2.2. Структура ПО

На рисунке 2.5 представлена диограмма классов.

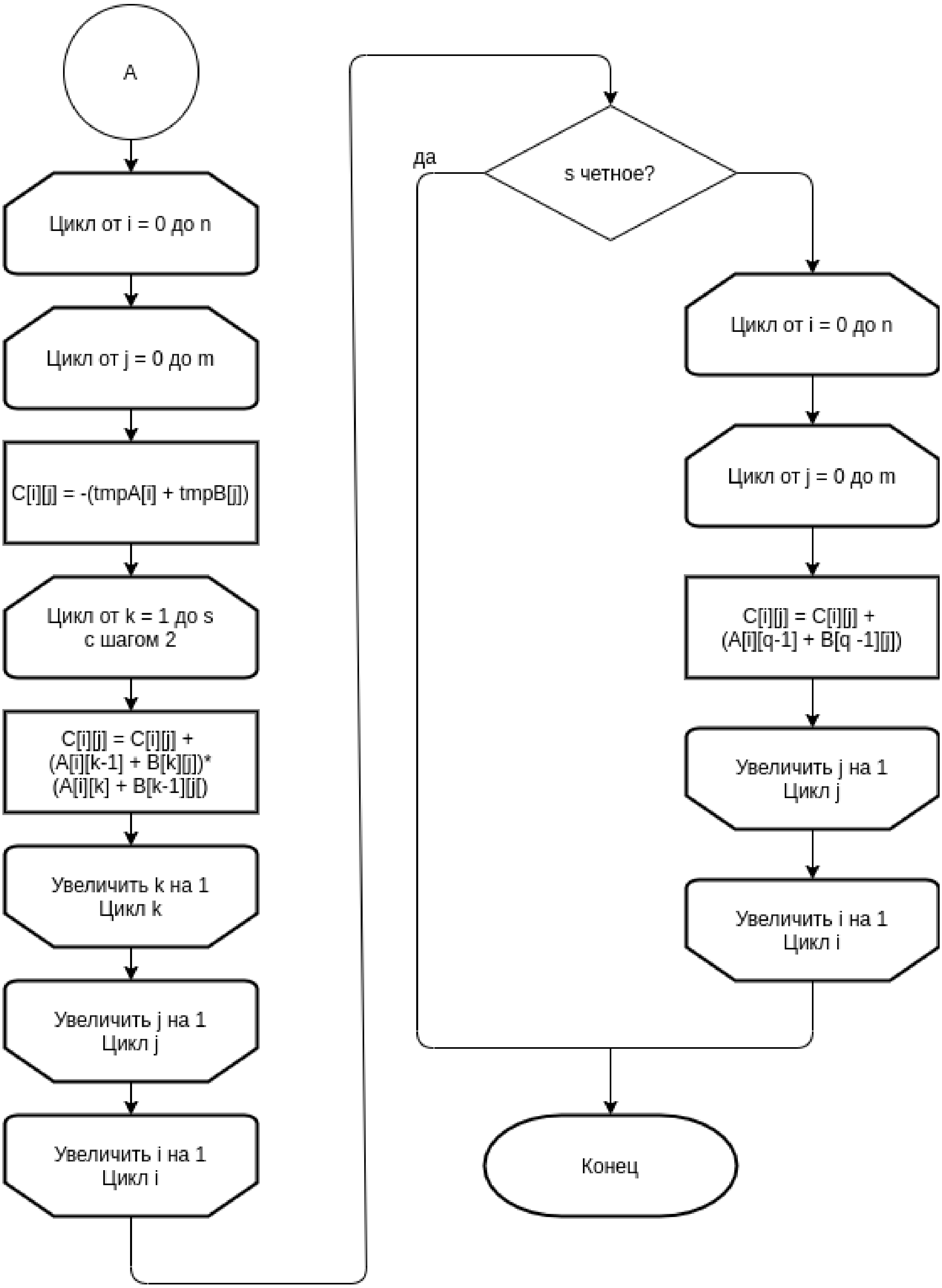

Рис. 2.3: Алгоритм Винограда умножения матриц(часть 2)

2.3. Тестирование

В рамках данной лабораторной работы были выделены следующие классы эквивалентности:

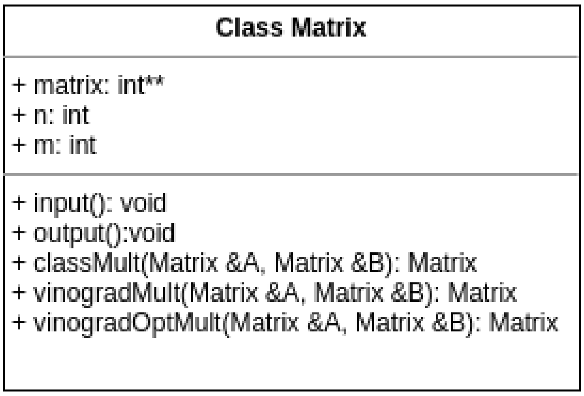

Рис. 2.4: Диаграмма классов реализуемого ПО

- входными данными являются две матрицы размерностью 1*1;
- входными данными являются две матрицы размерностью 1*n и n*1 соответственно;
- входными данными являются две матрицы квадратные матрицы равной размерностью;
- входными данными являются две матрицы размерностью n*k и к*т.

Для проверки работы программы будет осуществлено тестирование согласно классам эквивалентности.

Вывод

Были составлены схемы и подсчитана трудоёмкость для каждого алгоритма. Из последнего можно увидеть, что оптимизированный алгоритм Винограда менее трудоёмкий, чем неоптимизированный.

3. Технологическая часть

В данном разделе приведены средства реализации, требования к ПО и листинги кода.

3.1. Средства реализации

В качестве языка программирования был выбран с++. Данный язык знаком и предостовляет все необходимые ресурсы. В качестве среды разработки я использовала Visual Studio Code, т.к. считаю его достаточно удобным и легким. Visual Studio Code подходит не только для Windows, но и для Linux, это еще одна причина, по которой я выбрала VS code, т.к. у меня установлена ОС fedora 34.

3.2. Сведенья о модулях программы

- main.cpp - файл, содержащий точку входа в программу;
- matrix.cpp - файл, содержащий реализацию алгоритмов умножения матриц.

3.3. Реализация алгоритмов

В листингах 3.1 - 3.3 приведены реализации алгоритмов сортировки массивов на ЯП С++.

Листинг 3.1: Классический алгоритм умножения матриц.

```
void Matrix::classicMultiplication(Matrix &A, Matrix &B)
{
  for (int i = 0; i < this->n; i++)
  for (int j = 0; j < this->m; j++)
  for (int k = 0; k < A.column(); k++)
    this->matrix[i][j] += A.matrix[i][k] * B.matrix[k][j];
}
```

Листинг 3.2: Алгоритм Винограда.

```
void Matrix::vinogradMultiplication(Matrix &A, Matrix &B)
{
    precomputeRows(A);
```

```
    precomputeCols(B);

    for(int i = 0; i < A.row(); i++)
    for (int j = 0; j < B.column(); j++)
    {
        this->matrix[i][j] = -1*(tmpA[i], tmpB[j]);
        for (int k = 0; k < B.row()/2; k++)
        {
            this->matrix[i][j] = this->matrix[i][j] +
            (A.get(i, k*2) + B.get(k*2 + 1, j)) *
            (A.get(i, k*2 + 1) + B.get(k*2 , j));
        }
    }

    if (B.row()%2)
    {
        for (int i = 0 ; i < A.row(); i++)
            for (int j = 0; j < B.column(); j++)
            {
                this->matrix[i][j] = this->matrix[i][j] +
                A.get(i, B.row()/2 - 1)*
                B.get(B.row()/2 - 1, j);
        }
    }

    delete []tmpA;
    delete []tmpB;

}

void Matrix::precomputeRows(Matrix &A)
{
    tmpA = new int [A.row()];

```

```
    for (int i = 0; i < A.row(); i++)
        for (int j = 0; j < A.column()/2 ;j++)
        {
            tmpA[i] = tmpA[i] +
            A.get(i, j*2) * A.get(i, j*2 + 1);
        }
}

void Matrix::precomputeCols(Matrix &B)
{
    tmpB = new int [B.column()];

    for (int i = 0; i < B.column(); i++)
        for (int j = 0; j < B.row()/2 ;j++)
        {
             tmpB[i] = tmpB[i] +
              B.get(j*2, i) * B.get(j*2 + 1, i);
        }
}
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
void Matrix::vinogradOptMultipl(Matrix &A, Matrix &B)
{
    precomputeRowsOpt(A);
    precomputeColsOpt(B);

    int temp = 0;

    for(int i = 0; i < A.row(); i++)
    for (int j = 0; j < B.column(); j++)
    {
        temp = -1*(tmpA[i], tmpB[j]);
        for (int k = 0; k < B.row()/2; k++)
        {
            temp += (A.get(i, k-1) + B.get(k, j)) *
```

```
                (A.get(i, k) + B.get(k-1, j));
        }
        this->matrix[i][j] = temp;
    }

    if (B.row()%2)
    {
        for (int i = 0 ; i < A.row(); i++)
        for (int j = 0; j < B.column(); j++)
        {
            this->matrix[i][j] +=
            A.get(i, B.row() - 1) * B.get(B.row() - 1, j);
        }

    }

    delete []tmpA;
    delete []tmpB;
}


void Matrix::precomputeRowsOpt(Matrix &A)
{
    tmpA = new int [A.row()];

    for (int i = 0; i < A.row(); i++)
    {
        for (int j = 0; j < A.column() ;j+=2)
        {
            tmpA[i] = A.get(i, j-1) * A.get(i, j);
        }
    }
}

```

```
void Matrix::precomputeColsOpt(Matrix &B)
{
    tmpB = new int [B.column()];

    for (int i = 0; i < B.column(); i++)
    {
        for (int j = 0; j < B.row() ;j++)
        {
            tmpB[i] = B.get(j-1, i) * B.get(j, i);
        }
    }
}
```

3.4. Оценка трудоемкости

Для начала оценки алгоритмов, можно ввести специальную модель трудоёмкости:

- стоимость базовых операций 1 — +, -, *, /, =, == ...;
- оценка цикла — $f_{for} = f_{init} + N \cdot (f + f_{body} + f_{post}) + f$, где f - условие цикла, f_{init} - предусловие цикла, f_{post} - постусловие цикла;
- стоимость условного перехода примем за 0, стоимость вычисления условия остаётся.

Для стандартного алгоритма умножения с матрицами A и B и размерами $n \times m$ и $m \times l$ соответственно:

$$f = 2 + n \cdot (2 + 2 + l \cdot (2 + 2 + m \cdot (2 + 6 + 2))) = 10nlm + 4ln + 4n + 2$$

Для алгоритма Винограда при тех же матрицах и их размерах.

Для более понятного подсчёта, можно составить таблицу (таблица 2.1), а затем подсчитать общую трудоёмкость:

$$f = 13mnl+7.5mn+7.5lm+11ln+8n+4l+14+\begin{cases} 0, \text{если m чётное} \\ 15 \cdot l \cdot n + 4 \cdot n + 2, \text{иначе} \end{cases}$$

Таблица 3.1: трудоёмкость алгоритма Винограда

Часть алгоритма	Трудоёмкость
Инициализация mulH и mulV	$2 \cdot 3$
Заполнение mulH	$2 + n \cdot (2 + 2 + m/2 \cdot (3 + 6 + 6))$
Заполнение mulV	$2 + l \cdot (2 + 2 + m/2 \cdot (3 + 6 + 6))$
Подсчёт результата	$2 + n \cdot (2 + 2 + l \cdot (2 + 7 + 2 + m/2 \cdot (3 + 23)))$
Условный оператор нечёт. m	2
Для матриц с нечёт m	$2 + n \cdot (2 + 2 + l \cdot (2 + 8 + 5))$

Для оптимизированного алгоритма Винограда при тех же матрицах и размерах. Для более понятного подсчёта, можно тоже составить таблицу (таблица 2.2).

$$f = 8mnl + 5mn + 5lm + 12ln + 8n + 4l + 18 + \begin{cases} 0, \text{если m чётное} \\ 10 \cdot l \cdot n + 4 \cdot n + 4, \text{иначе} \end{cases}$$

Таблица 3.2: трудоёмкость оптимизированного алгоритма Винограда

Часть алгоритма	Трудоёмкость
Инициализация mulH и mulV	$2 \cdot 3$
Инициализация m1Mod2 и n2Mod2	$2 \cdot 2$
Заполнение mulH	$2 + n \cdot (2 + 2 + m/2 \cdot (2 + 5 + 3))$
Заполнение mulV	$2 + l \cdot (2 + 2 + m/2 \cdot (2 + 5 + 3))$
Подсчёт результата	$2 + n \cdot (2 + 2 + l \cdot (2 + 5 + 3 + 2+$ $+m/2 \cdot (2 + 14)))$
Условный оператор нечёт. m	2
Для матриц с нечёт m	$2 + 2 + n \cdot (2 + 2 + l \cdot (2 + 6 + 2))$

3.5. Тестирование

В данном разделе будет приведена таблица с тестами (таблица 3.3).

Таблица 3.3: Таблица тестов

Матрица A	Матрица B	Результат
2 2 1 0 0 1	2 2 1 0 0 1	Ответ верный
3 2 2 3 1 0 2 2	2 4 2 2 1 9 4 2 8 1	Ответ верный
2 1 2 1	12 12	Ответ верный
2 2 1 0 0 1	1 1 0	Сообщение о неверном вводе размерностей
0 0	0 0	

Все тесты пройдены.

Вывод

В данном разделе были приведены средства реализации, требования к ПО и листинги кода.

4. Исследовательская часть

В данном разделе сравним работу каждого алгоритма.

4.1. Временные характеристики

Для сравнения возьмем квадратные матрицы размерностью [10, 20, 30,…,100]. Так как подсчет умножения матриц считается короткой задачей, воспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма n раз (n >= 10), после чего поделим на n. Тем самым получим достаточно точные характеристики времени. Сравнение произведем при n = 50. Результат можно увидеть на рис. 4.1.

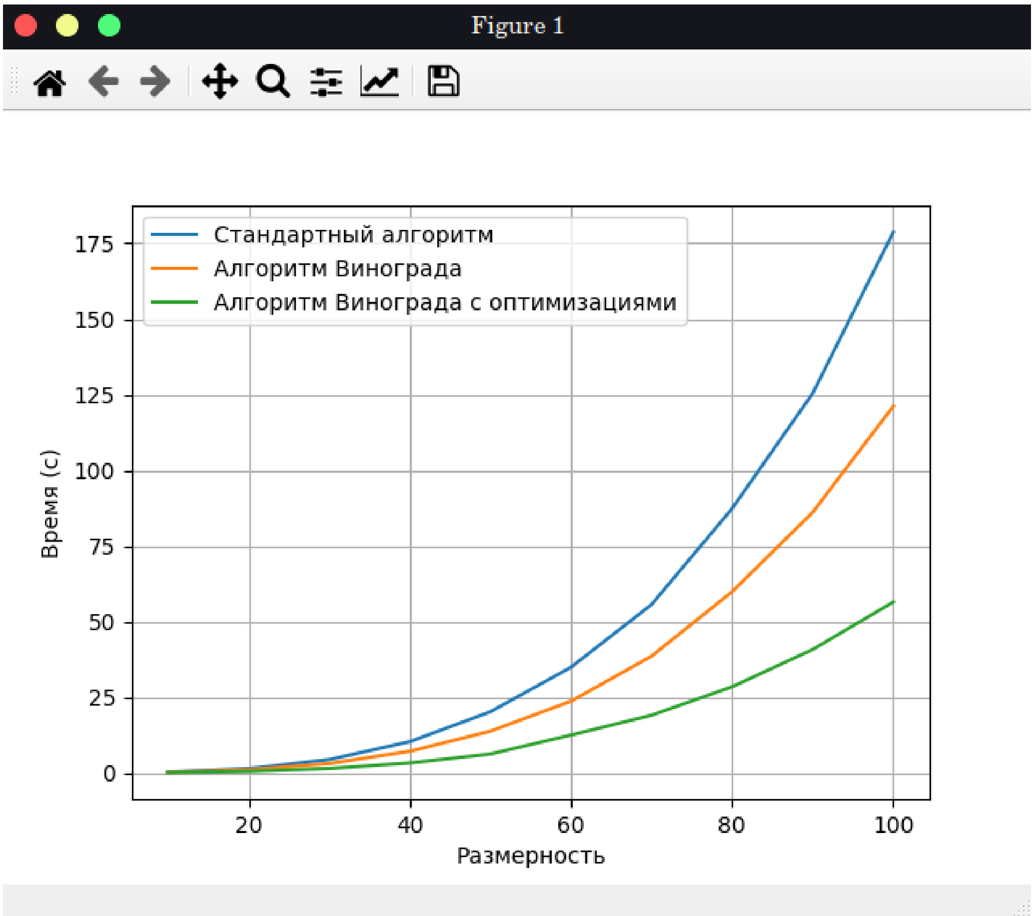

Рис. 4.1: Временные характеристики на четных размерах матриц

На рис. 4.2 показана работа алгоритмов с матрицами, размерностью [11, 21, 31,…,91].

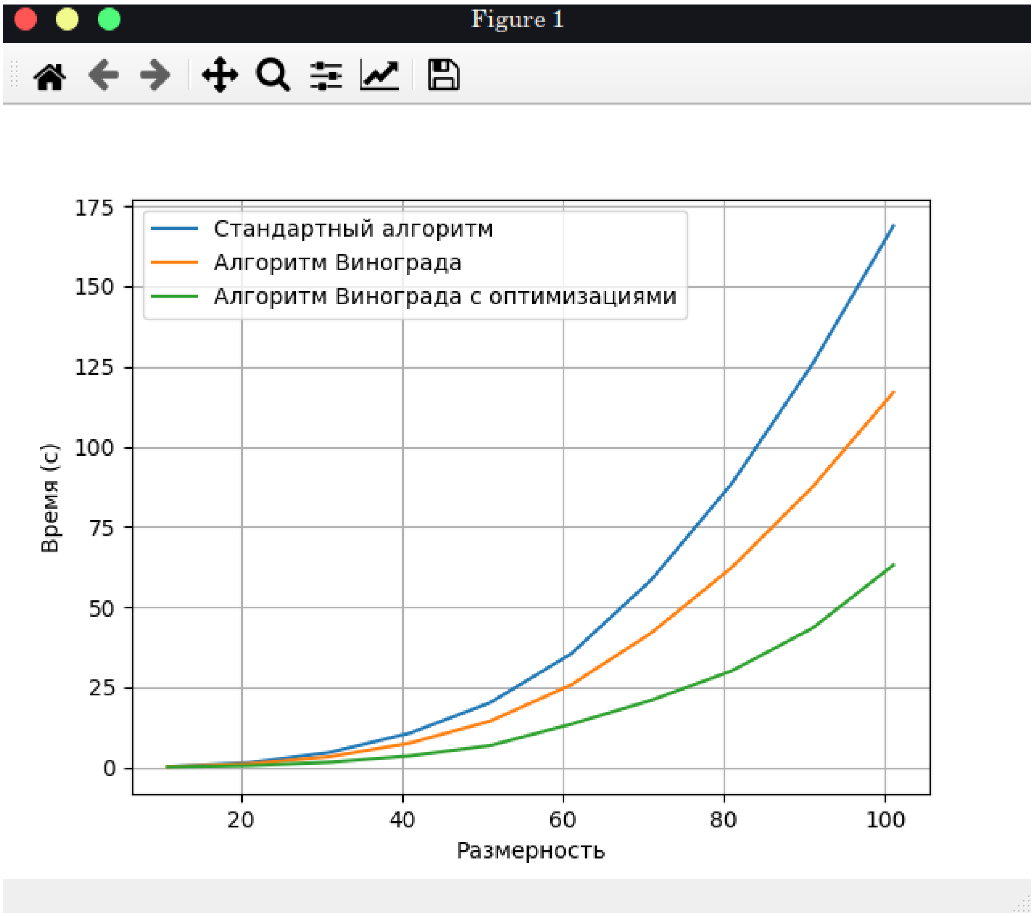

Рис. 4.2: Временные характеристики на нечетных размерах матриц

4.2. Сравнительный анализ алгоритмов

Введем модель вычислений трудоемкости алгоритма. Пусть трудоемкость 1 у следующих базовых операций: +, -, *, /, %, =, ==, !=, <, <=, >, >=, []. fинкрем + fсравн).Трудоемкость условного перехода 1.

Стандартная реализация алгоритма не эффективна по времени, так как обладает трудоемкостью 5qmn + 4n + 4mn + 5. Оценка трудоемкости данного алгоритма составляет 5qmn. По памяти в стандартном алгоритме умножения матриц требуется m*n памяти под результат.

Теперь рассмотрим алгоритм Винограда умножения матриц. Реализация алгоритма Винограда обладает трудоемкостью формула 4.1. Оценка трудоемкости данного алгоритма составляет 3qmn. В алгоритме Винограда умножения матриц требуется дополнительно m+n памяти под результат.

$$3qmn + 7mn + 2m + 5q + 6n + 11 + \left[\begin{array}{c} 0 \text{ л.с.} \\ 5mn + 4n + 2 \text{ х.с.} \end{array} \right. \quad (4.1)$$

Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Самым быстрым оказался модифицированный алгоритм Винограда. При этом в алгоритме Винограда умножения матриц требуется дополнительно m+n памяти под результат.

Заключение

В ходе работы были изучены алгоритмы умножения матриц. Реализованы 3 алгоритма, приведен программный код реализации алгоритмов по умножению матриц. Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение алгоритмов по времени и трудоемкости. Показано, что наименее трудоемкий и наименее затратный по времени при больших размерах матриц алгоритм Винограда оптимизированный.

Цель работы достигнута, решены поставленные задачи. Получены практические навыки реализации алгоритмов Винограда и стандартного алгоритма, а также проведена исследовательская работа по оптимизации и вычислении трудоемкости алгоритмов.

Список литературы

1. Дж. Макконнел. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с.

2. Основы программирования на языках Си и C++ для начинающих[Электронный ресурс]. Режим доступа: http://cppstudio.com/ (дата обращения 10.10.2021)

3. LINUX.ORG.RU - Русскоязычная информация о ОС Linux[Электронный ресурс] Режим доступа://www.linux.org.ru/(дата обращения 25.10.2021)

4. Погорелов Д. А. Таразанова А. М. Волкова Л. Л. Оптимизация классического алгоритма Винограда для перемножения матриц // Журнал №1 2019. Т. 49.

5. Корн Г., Корн Т. Алгебра матриц и матричное исчисление // Справочник по математике. — 4-е издание. — М.: Наука, 1978. — С. 392—394.