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»

Отчет по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Алгоритм Копперсмита-Винограда

Студент Якуба Д. В.

Группа ИУ7-53Б

Оценка (баллы) ____________

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Оглавление

Введение

Цели лабораторной работы

1. изучение алгоритмов умножения матриц: классического, Копперсмита-Винограда и модифицированного Копперсмита-Винограда;
2. реализация алгоритмов умножения матриц: классического, Копперсмита-Винограда и модифицированного Копперсмита-Винограда;
3. проведение сравнительного анализа трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
4. сравнительный анализ алгоритмов на основе экспериментальных данных;
5. подготовка отчёта по лабораторной работе.

Определение

Алгоритм Копперсмита-Винограда - это алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом [1]. В исходной весрии асимптотическая сложность алгоритма составляла $O(n^{2.3755})$, где n - это размер стороны матрицы. Алгоритм Копперсмита-Винограда с учётом усерии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров [2]. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости [3].

1 Аналитическая часть

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы

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

тогда матрица C

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

где

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

будет называться произведением матриц A и B.

Реализация классического алгоритма умножения двух матриц заключается в реализации вычисления элементов итоговой матрицы по формуле 1.3

1.2 Алгоритм Копперсмита-Винограда умножения матриц

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 равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

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

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: 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 позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного [4].

Вывод

Были рассмотрены классический алгоритм множения матриц и алгоритм Копперсмита-Винограда умножения матриц. Основное отличие данных алгоритмов заключается в наличии предварительной обработки данных и количестве проводящихся операций умножения.

2 Конструкторская часть

2.1 Блок-схема классического алгоритма умножения матриц

Блок-схема классического алгоритма умножения матриц предоставлена на рисунке 2.1

2.2 Блок-схема алгоритма Копперсмита-Винограда

Блок-схема алгоритма Копперсмита-Винограда и алгоритмов предвычисления предоставлены на рисунках 2.2 - 2.5.

2.3 Блок-схема улучшенного алгоритма Копперсмита-Винограда

Блок-схема улучшенного алгоритма Копперсмита-Винограда и алгоритмов предвычисления предоставлены на рисунках 2.6 - 2.9.

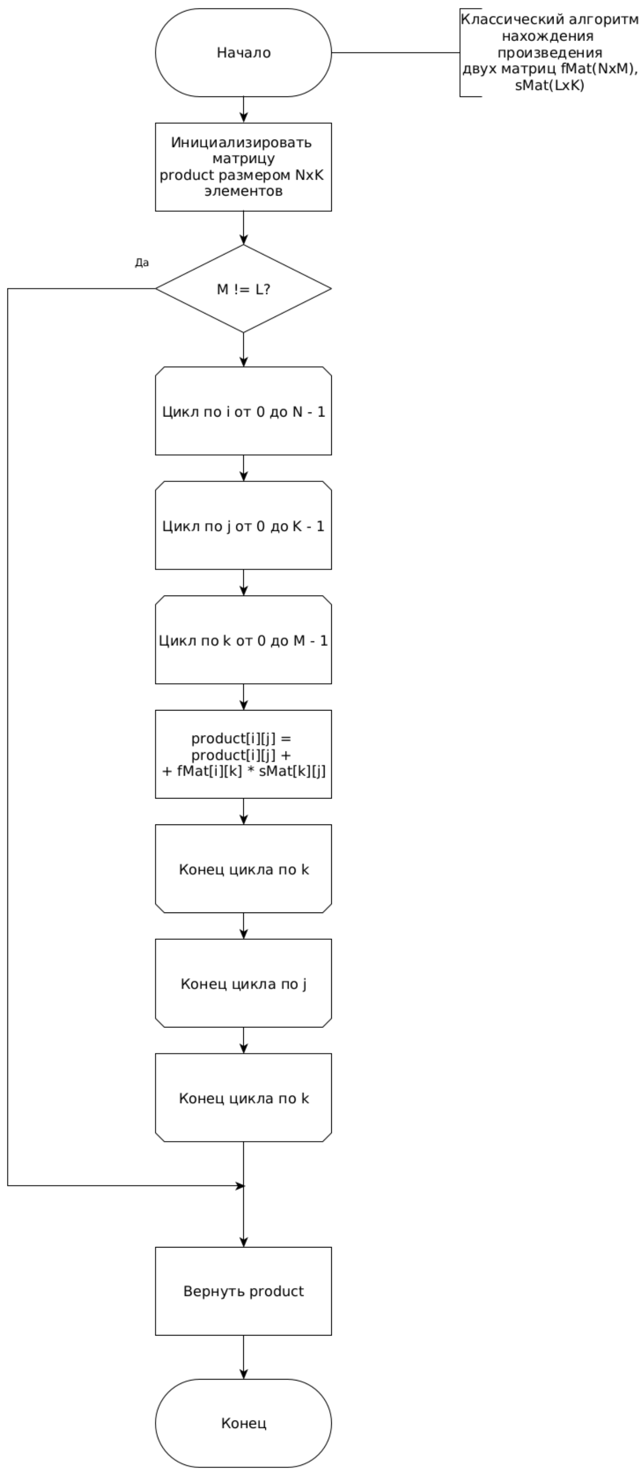

Рис. 2.1: Блок-схема классического алгоритма умножения матрицы.

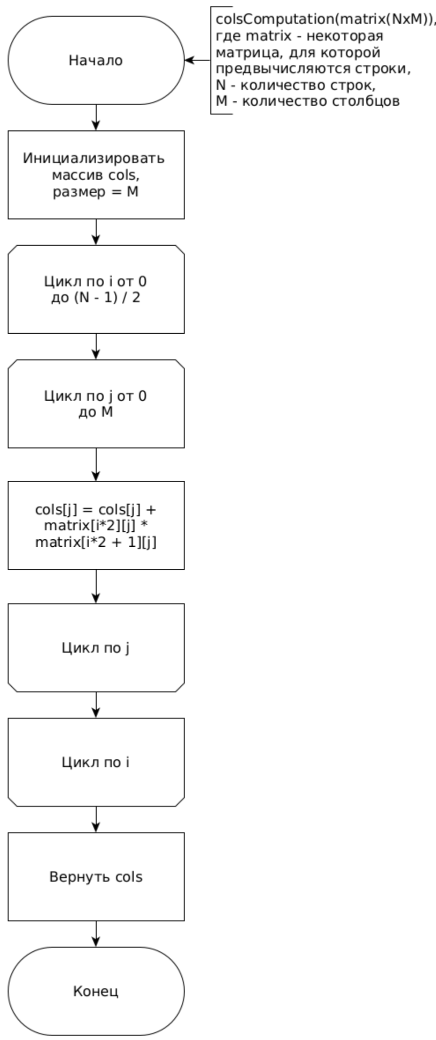

Рис. 2.2: Блок-схема алгоритма предвычисления столбцов.

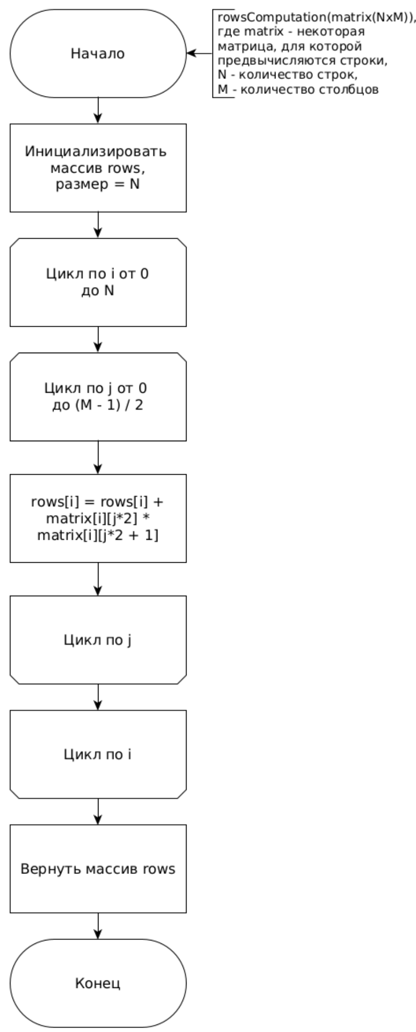

Рис. 2.3: Блок-схема алгоритма предвычисления строк.

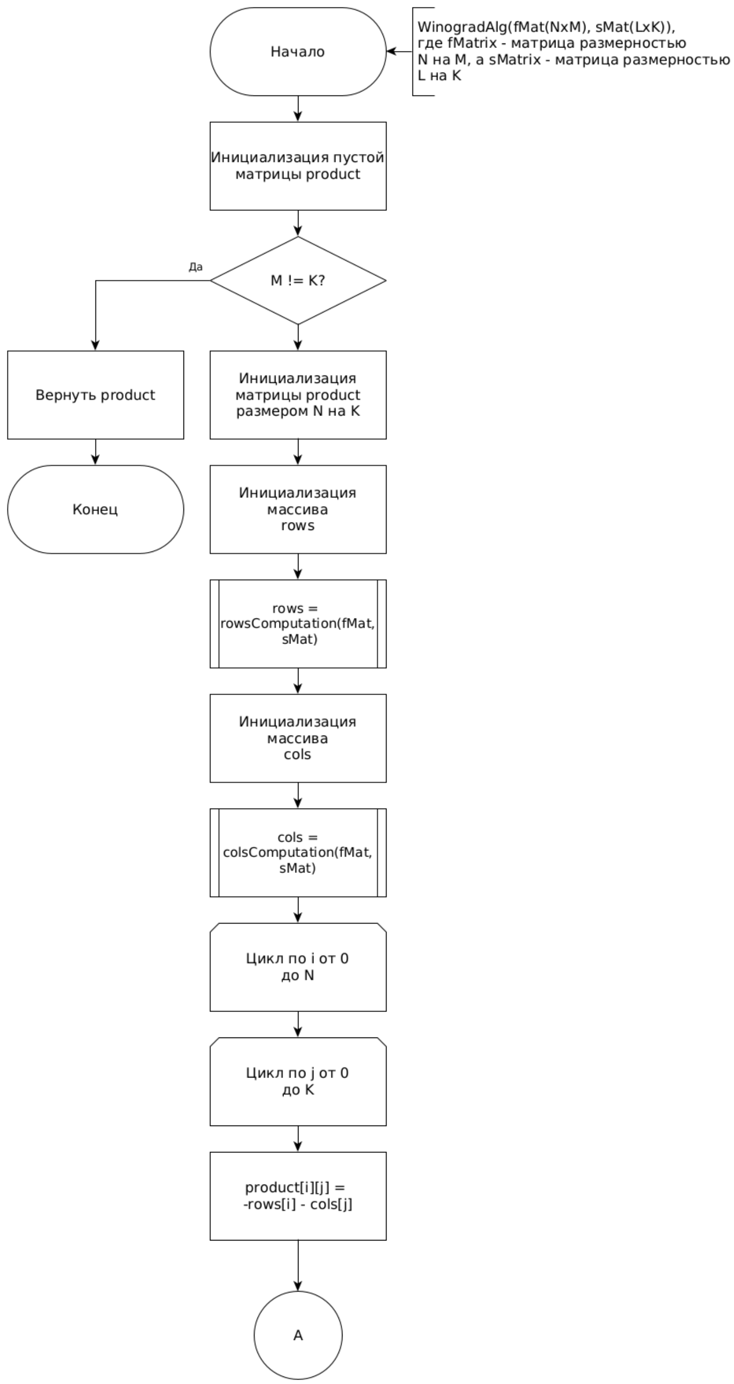

Рис. 2.4: Блок-схема алгоритма Копперсмита-Винограда.

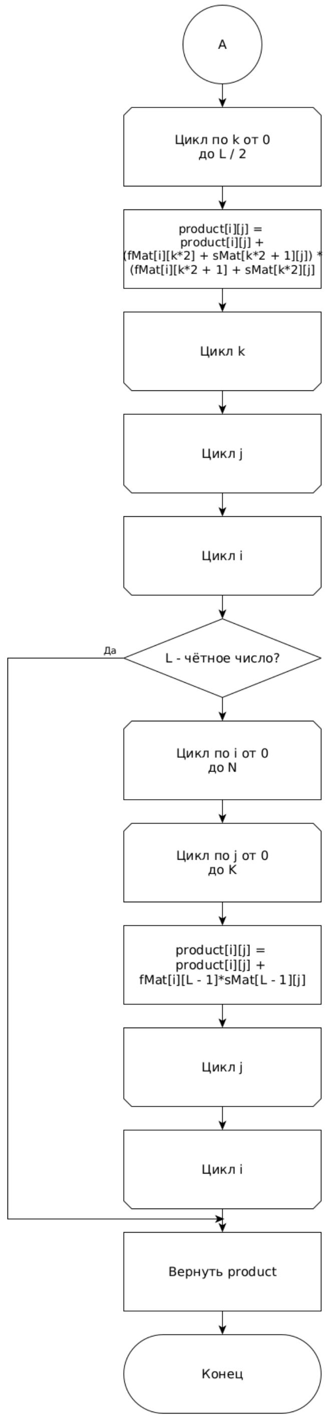

Рис. 2.5: Продолжение блок-схемы алгоритма Копперсмита-Винограда.

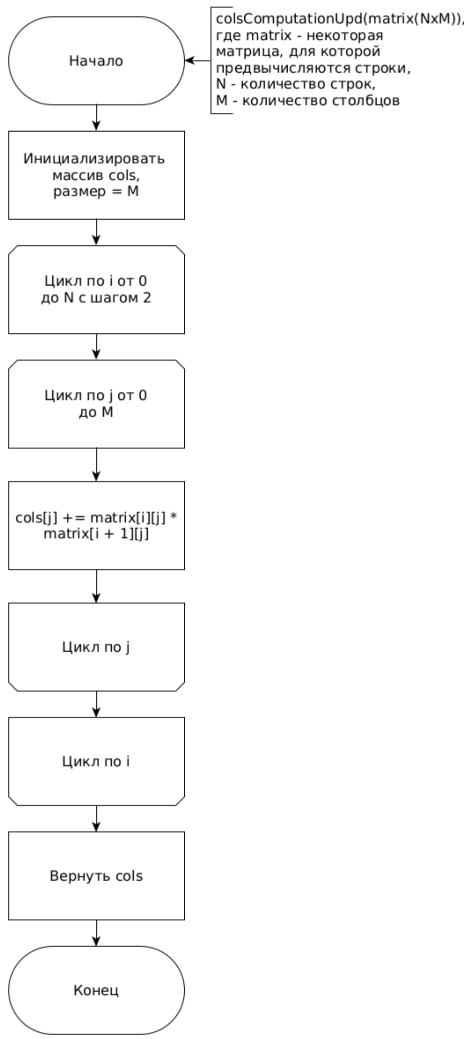

Рис. 2.6: Блок-схема алгоритма предвычисления столбцов.

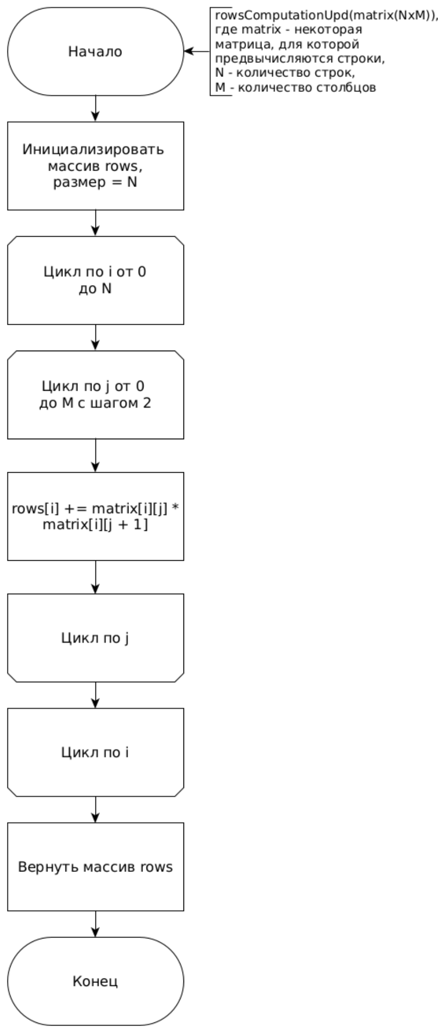

Рис. 2.7: Блок-схема алгоритма предвычисления строк.

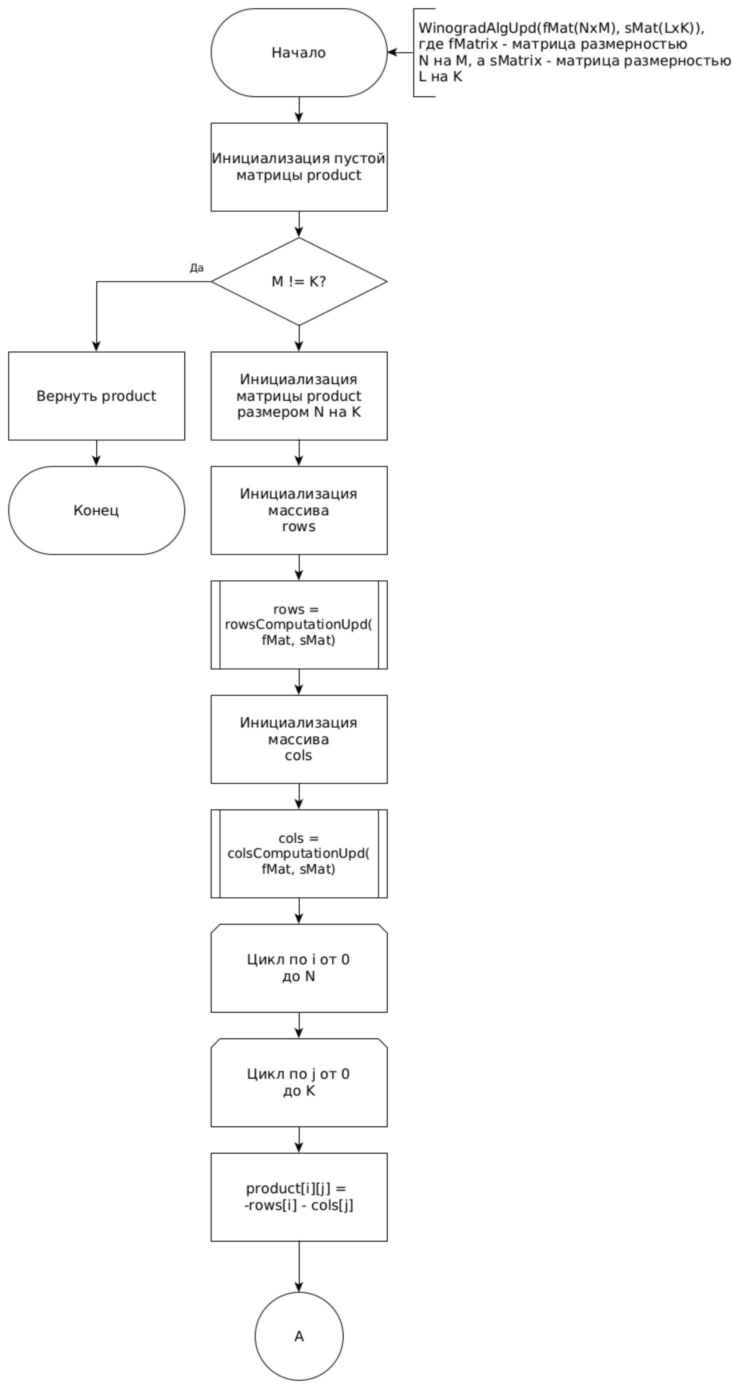

Рис. 2.8: Блок-схема улучшенного алгоритма Копперсмита-Винограда.

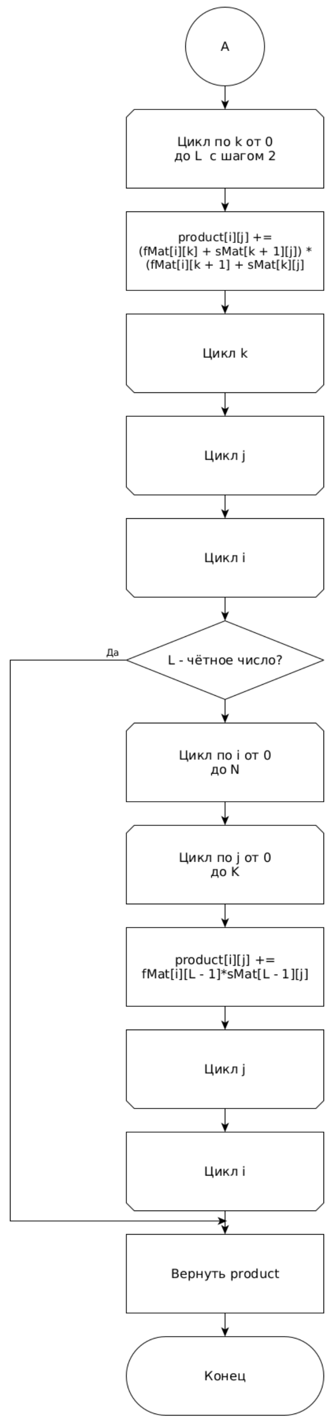

Рис. 2.9: Продолжение блок-схемы улучшенного алгоритма Копперсмита-Винограда.

2.4 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, \%, ==, != , <, >, <=, >=, [], ++, --, +=, -= \tag{2.1}$$

2. трудоемкость оператора выбора **условие then A else B** рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. трудоемкость цикла рассчитывается, как (**??**);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.3}$$

4. трудоемкость вызова функции равна 0.

2.5 Трудоёмкость алгоритмов

2.5.1 Классический алгоритм

Трудоёмкость стандартного алгоритма умножения двух матриц будет включать в себя:

- Цикл по $i \in [1..N]$, где N - количество строк первой переданной матрицы;
- Цикл по $j \in [1..K]$, где K - количество столбцов второй переданной матрицы;
- Цикл по $k \in [1..M]$, где M - количество столбцов первой переданной матрицы.

При этом для цикла по $i \in [1..N]$ будем иметь трудоёмкость f_i такую, что: $f_i = 2 + N \cdot (2 + f_{body})$, где f_{body} - трудоёмкость тела цикла.

Для цикла по $j \in [1..K]$ трудоёмкость $f_j = 2 + K \cdot (2 + f_{body})$.

Для цикла по $k \in [1..M]$ трудоёмкость $f_k = 2 + 8 \cdot M$.

Как итог, общая трудоёмкость алгоритма может быть представлена выражением 2.4:

$$f = 2 + N \cdot (2 + (2 + K \cdot (2 + 2 + 8 \cdot M))) \tag{2.4}$$

При этом выражение 2.4 преобразуется в 2.5:

$$f = 2 + 4 \cdot N + 4 \cdot K \cdot N + 8 \cdot N \cdot K \cdot M \tag{2.5}$$

2.5.2 Алгоритм Копперсмита-Винограда

Трудоёмкость алгоритма Копперсмита-Винограда умножения двух матриц будет включать в себя:

- Создание и инициализация массивов $cols$ и $rows$;
- Заполнение массива $cols$;
- Заполнение массива $rows$;
- Цикл заполнения итоговой матрицы для чётных размеров матриц;
- Цикл для дополнительного заполнения при нечётных размеров матриц;

Трудоёмкость создания и инициализации массивов $cols$ и $rows$: $f_{rc} = N + M$, где M - размер массива cols, количество столбцов второй переданной матрицы, а N - размер массива $rows$, количество строк первой переданной матрицы.

Трудоёмкость заполнения массива $cols$: $f_{cols} = 5 + \frac{M-1}{2}(2 + 11L)$, где M - количество строк второй переданной матрицы, L - количество столбцов второй переданной матрицы.

Трудоёмкость заполнения массива $rows$: $f_{rows} = 2 + N(5 + \frac{M-1}{2} \cdot 11)$, где N - количество строк первой переданной матрицы, M - количество столбцов первой переданной матрицы.

Трудоёмкость цикла заполнения итоговой матрицы для чётных размеров матриц: $f_{evencyc} = 2 + L(2 + M(9 + 11N))$, где N - количество строк первой матрицы, M - количество столбцов первой матрицы, L - количество столбцов второй матрицы.

Трудоёмкость цикла для дополнительного заполнения при нечётных размерах матриц: $f_{end} = 2 + N(2 + 12L)$. При этом рассматривается два случая, представленных в выражении 2.6:

$$f_{end} = \begin{cases} 2, & \text{чётная,} \\ 4 + N(2 + 12L), & \text{иначе.} \end{cases} \tag{2.6}$$

Общую формулу для вычисления тудоёмкости можно представить как 2.7:

$$f = f_{rc} + f_{cols} + f_{rows} + f_{evencyc} + f_{end} \tag{2.7}$$

Для худшего случая, когда размеры матриц нечётные, из 2.7:

$$f = 12+8N+11N\frac{M-1}{2}+2M+11L\frac{M-1}{2}+9NL+11N^2L+2N+12NL \approx 11N^2L \tag{2.8}$$

Для лучшего случая, когда размеры матриц чётные, из 2.7:

$$f = 10+8N+11N\frac{M-1}{2}+2M+11L\frac{M-1}{2}+9NL+11N^2L \approx 11N^2L \tag{2.9}$$

2.5.3 Улучшенный алгоритм Копперсмита-Винограда

Трудоёмкость улучшенного алгоритма Копперсмита-Винограда умножения двух матриц будет включать в себя:

- Создание и инициализация массивов $cols$ и $rows$;
- Заполнение массива $cols$;
- Заполнение массива $rows$;
- Цикл заполнения итоговой матрицы для чётных размеров матриц;
- Цикл для дополнительного заполнения при нечётных размеров матриц;

Трудоёмкость создания и инициализации массивов $cols$ и $rows$: $f_{rc} = N + M$, где M - размер массива cols, количество столбцов второй переданной матрицы, а N - размер массива $rows$, количество строк первой переданной матрицы.

Трудоёмкость заполнения массива $cols$: $f_{cols} = 4 + \frac{M-1}{2}(2+8L)$, где M - количество строк второй переданной матрицы, L - количество столбцов второй переданной матрицы.

Трудоёмкость заполнения массива $rows$: $f_{rows} = 2 + N(4 + \frac{M-1}{2} \cdot 8)$, где N - количество строк первой переданной матрицы, M - количество столбцов первой переданной матрицы.

Трудоёмкость цикла заполнения итоговой матрицы для чётных размеров матриц: $f_{evencyc} = 2 + L(2 + M(8 + 8N))$, где N - количество строк первой матрицы, M - количество столбцов первой матрицы, L - количество столбцов второй матрицы.

Трудоёмкость цикла для дополнительного заполнения при нечётных размерах матриц: $f_{end} = 2 + N(2 + 10L)$. При этом рассматривается два случая, представленных в выражении 2.6:

$$f_{end} = \begin{cases} 2, & \text{чётная,} \\ 4 + N(2 + 10L), & \text{иначе.} \end{cases} \tag{2.10}$$

Общую формулу для вычисления тудоёмкости можно представить как 2.11:

$$f = f_{rc} + f_{cols} + f_{rows} + f_{evencyc} + f_{end} \tag{2.11}$$

Для худшего случая, когда размеры матриц нечётные, из 2.11:

$$f = M+N+4+\frac{M-1}{2}(2+8L)+2+N(4+\frac{M-1}{2}\cdot8)+2+L(2+M(8+8N))+4+N(2+10L) \approx 8MLN \tag{2.12}$$

Для лучшего случая, когда размеры матриц чётные, из 2.11:

$$f = M+N+4+\frac{M-1}{2}(2+8L)+2+N(4+\frac{M-1}{2}\cdot8)+2+L(2+M(8+8N))+2 \approx 8MLN \tag{2.13}$$

Вывод

На основе теоретичесих данных, полученных из аналитического раздела, были построены схемы рассматриваемых алгоритмов умноежния матриц, оценены их трудоёмкости в лучшем и худшем случаях.

3 Технологическая часть

3.1 Требования к программному обеспечению

- Входные данные - две матрицы размерностью MxN и KxL;
- Выходые данные - результат умножения двух переданных матриц.

3.2 Средства реализации программного обеспечения

При написании программного продукта был использован язык программирования Kotlin [5].

Данный выбор обусловлен следующими факторами:

- Высокая вычислительная производительность;
- Большое количество справочной литературы, связанной с ЯП Java.

Для тестирования производительности реализаций алгоритмов использовалась утилита measureNanoTime.

При написаннии программного продукта использовалась среда разработки IntelliJ IDEA.

Данный выбор обусловлен тем, что язык программирования Kotlin - это разработка компании JetBrains, поставляющей данную среду разработки.

3.3 Листинг кода

В листингах 3.1 - 3.3 предоставлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Функция реализации алгоритма классического умножения матриц

```
fun matricesMult(fMatrix: Array<IntArray>, sMatrix: Array<IntArray>) : Array<IntArray>
{
    if (fMatrix[0].size != sMatrix.size)
        return emptyArray()

    val product = Array(fMatrix.size) { IntArray(sMatrix[0].size) }
    for (i in fMatrix.indices)
        for (j in sMatrix[0].indices)
            for (k in fMatrix[0].indices)
                product[i][j] += fMatrix[i][k] * sMatrix[k][j]
    return product
}
```

Листинг 3.2: Функция реализации алгоритма Копперсмита-Винограда

```
fun rowsComputation(matrix: Array<IntArray>) : IntArray
{
    val computedRows = IntArray(matrix.size)

    for (i in matrix.indices)
        for (j in 0 until (matrix[0].size - 1) / 2)
            computedRows[i] = computedRows[i] + matrix[i][j * 2] * matrix[i][j * 2 + 1]

    return computedRows
}

fun colsComputation(matrix: Array<IntArray>) : IntArray
{
    val computedCols = IntArray(matrix[0].size)

    for (i in 0 until (matrix.size - 1) / 2)
        for (j in matrix[0].indices)
            computedCols[j] = computedCols[j] + matrix[i * 2][j] * matrix[i * 2 + 1][j]
```

```

    return computedCols
}

fun WinogradMultiplication(fMatrix: Array<IntArray>, sMatrix: Array<IntArray>) : Array<IntArray>
{
    if (fMatrix[0].size != sMatrix.size)
        return emptyArray()

    val computedRows = rowsComputation(fMatrix)
    val computedCols = colsComputation(sMatrix)

    val product = Array(fMatrix.size) { IntArray(sMatrix[0].size) }
    for (i in product.indices)
        for (j in product[0].indices)
        {
            product[i][j] = -computedRows[i] - computedCols[j]

            for (k in 0 until (sMatrix.size / 2))
                product[i][j] = product[i][j] + (fMatrix[i][k * 2] + sMatrix[k * 2 + 1][j]) * (fMatrix[i][k * 2 + 1] + sMatrix[k * 2][j])
        }

    if (sMatrix.size % 2 != 0)
    {
        for (i in product.indices)
            for (j in product[0].indices)
                product[i][j] = product[i][j] + fMatrix[i][sMatrix.size - 1] * sMatrix[sMatrix.size - 1][j]
    }

    return product
}
```

Листинг 3.3: Функция реализации улучшенного алгоритма Копперсмита-Винограда

```
fun rowsComputationModified(matrix: Array<IntArray>) : IntArray
{
    val computedRows = IntArray(matrix.size)
    for (i in matrix.indices)
```

```
        for (j in matrix[0].indices step 2)
            computedRows[i] += matrix[i][j] * matrix[i][j + 1]

    return computedRows
}

fun colsComputationModified(matrix: Array<IntArray>) : IntArray
{
    val computedCols = IntArray(matrix[0].size)

    for (i in matrix.indices step 2)
        for (j in matrix[0].indices)
            computedCols[j] += matrix[i][j] * matrix[i + 1][j]

    return computedCols
}

fun WinogradMultiplicationModified(fMatrix: Array<IntArray>,
    sMatrix: Array<IntArray>) : Array<IntArray>
{
    if (fMatrix[0].size != sMatrix.size)
        return emptyArray()

    val computedRows = rowsComputationModified(fMatrix)
    val computedCols = colsComputationModified(sMatrix)

    val product = Array(fMatrix.size) { IntArray(sMatrix[0].size)
        }
    for (i in product.indices)
        for (j in product[0].indices)
        {
            product[i][j] = -computedRows[i] - computedCols[j]

            for (k in 0 until (sMatrix.size - 1) step 2)
                product[i][j] += (fMatrix[i][k] + sMatrix[k + 1][
                    j]) * (fMatrix[i][k + 1] + sMatrix[k][j])
        }

    if (sMatrix.size % 2 != 0)
    {
        for (i in product.indices)
            for (j in product[0].indices)
                product[i][j] += fMatrix[i][sMatrix.size - 1] *
                    sMatrix[sMatrix.size - 1][j]
    }
```

```

    return product
}
```

3.4 Тестирование программного продукта

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Копперсмита-Винограда и оптимизированный алгоритм Копперсмита-Винограда. Тесты пройдены успешно.

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 6 & 15 & 15 \\ 6 & 15 & 15 \\ 3 & 8 & 8 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 4 \\ 1 & 2 & 4 \end{pmatrix}$	$\begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 32 \\ 32 \end{pmatrix}$
$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 666 \end{pmatrix}$	$\begin{pmatrix} 3330 \end{pmatrix}$
$\begin{pmatrix} -1 & -2 & 3 \\ 1 & 2 & 3 \\ -1 & -2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 4 \\ 14 & 14 & 14 \\ 4 & 4 & 4 \end{pmatrix}$
$\begin{pmatrix} 666 & 666 \end{pmatrix}$	$\begin{pmatrix} 777 & 777 \end{pmatrix}$	Ошибка

Таблица 3.1: Тестирование функций

Вывод

Спроектированные алгоритмы вычисления произведения двух матриц были реализованы и протестированы.

4 Исследовательская часть

4.1 Пример работы программного обеспечения

Ниже на рисунках 4.1- 4.2 предоставлены примеры работы каждого из алгоритмов на случайных данных, сгенерированных один раз, и введённых пользователем данных:

```
First matrix is:
    11     1     4
   -10   -16    -5
    14    12   -10
Second matrix is:
   -17   -10    11
   -12    -8   -17
   -14     8   -18



Result of multiplication in classic:
  -255   -86    32
   432   188   252
  -242  -316   130



Result of multiplication in Winograd
  -255   -86    32
   432   188   252
  -242  -316   130



Result of multiplication in Upd Winograd
  -255   -86    32
   432   188   252
  -242  -316   130

Process finished with exit code 0
```

Рис. 4.1: Пример работы ПО.

```
Rows number:
3
Cols number:
3
Matrix elems:
1 2 3
2 2 2
3 3 3

Rows number:
3
Cols number:
3
Matrix elems:
1 2 3
3 2 1
9 6 6
First matrix is:
     1     2     3
     2     2     2
     3     3     3
Second matrix is:
     1     2     3
     3     2     1
     9     6     6


Result of multiplication in classic:
    34    24    23
    26    20    20
    39    30    30


Result of multiplication in Winograd
    34    24    23
    26    20    20
    39    30    30


Result of multiplication in Upd Winograd
    34    24    23
    26    20    20
    39    30    30
```

Рис. 4.2: Пример работы ПО.

4.2 Технические характеристики

Технические характеристики ЭВМ, на котором выполнялись исследования:

- ОС: Manjaro Linux 20.1.1 Mikah
- Оперативная память: 16 Гб
- Процессор: Intel Core i7-10510U

  При проведении замеров времени ноутбук был подключен к сети электропитания.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались на данных, сгенерированных случайным образом один раз.

Результаты замеров времени приведены в таблице 4.1. На рисунках 4.3 и 4.4 приведены графики зависимостей времени работы алгоритмов от количества строк и столбцов матриц (в чётном и нечётном вариантах). В таблице КА - Классический Алгоритм, КВ - Алгоритм Копперсмита-Винограда, УКВ - Улучшенный Алгоритм Копперсмита-Винограда.

Таблица 4.1: Замеры времени для квадратных матриц различных размеров

Размер матрицы	КА	КВ	УКВ
100	2148121	2302331	1921005
101	2312114	2623891	2032155
200	17350292	22592034	1425033
201	20247410	21694235	17554153
300	68920554	73453362	57000923
301	75166547	77955778	64421195
400	211301483	205981760	172968826
401	227614782	218087162	171881527
500	367822853	351730341	340284336
501	364368768	362588416	358108198
600	678478122	658012453	625149992
601	672846913	671159157	647843183

Вывод

При сравнении результатов замеров времени заметно, что скорость работы классического алгоритма однозначно отстаёт от скорости работы улучшенного Алгоритма Копперсмита-Винограда. Уже на 600 элементах улучшенный алгоритм Копперсмита-Винограда работает быстрее классического на $\approx 8\%$. При нечётном количестве строк и столбцов матриц улучшенный алгоритм способен быть медленнее $\approx 4\%$, при факте того, что классический алгоритм похожей динамики не имеет. Обычный алгоритм Копперсмита-Винограда начинает выигрывать по скорости классический только по достижению 300 строк и столбцов в матрице, при факте того, что в случае матрицы с нечётной размерностью он всё ещё будет проигрывать. При размерности 600 он будет выгрывать у классической реализации на $\approx 3\%$. В случае матриц размера меньше 400 на 400 его использование не будет целесообразным.

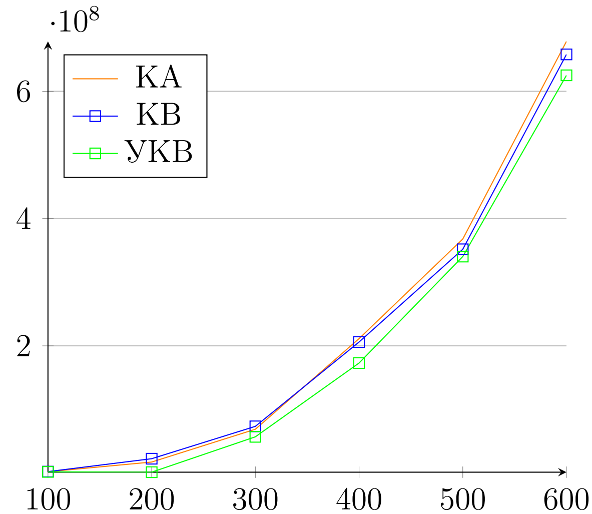

Рис. 4.3: Зависимость времени работы от размера матриц (чётные значения размерностей)

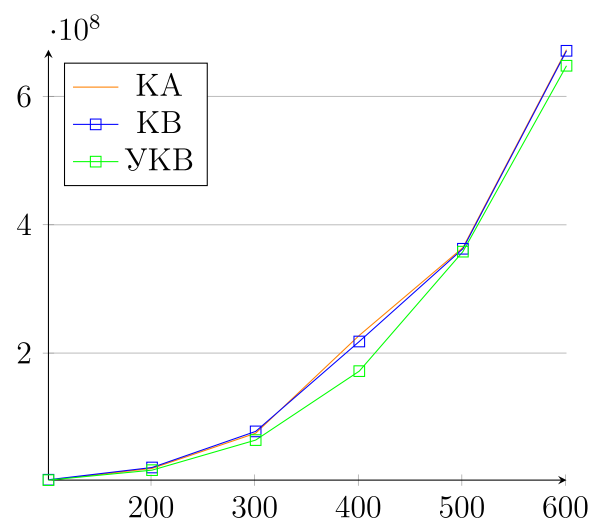

Рис. 4.4: Зависимость времени работы от размера матриц (нечётные значения размерностей)

Заключение

В ходе выполнения лабораторной работы:

- были изучены алгоритмы умножения матриц: классический, Копперсмита-Винограда и улучшенный Копперсмита-Винограда;
- были реализованы алгоритмы умножения матриц: классический, Копперсмита-Винограда и улучшенный Копперсмита-Винограда;
- был произведён анализ трудоёмкости указанных алгоритмов на основе теоретических расчётов и выбранной модели вычислений;
- был выполнен сравнительный анализ производительности алгоритмов на основе полученных экспериментальных данных;
- был подготовлен отчёт по проделанной работе;
- Были получены практические навыки реализации алгоритмов на ЯП Kotlin.

Исследования показали, что использование алгоритма Копперсмита-Винограда способно оправдать себя только в случае матриц, размерность которых не менее 400. При этом выигрыш будет составлять $\approx 0.02\%$ только в случае чётной размерности. Реализация улучшенного алгоритма Копперсмита-Винограда показывает быстрее классического алгоритма уже при размерности матрицы 100. Чем больше элементов в матрице - тем заметнее разница во времени работы этих двух алгоритмов.

Литература

[1] Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions // Journal of Symbolic Computation. 1990. no. 9. P. 251–280.

[2] Robinson S. Toward an Optimal Algorithm for Matrix Multiplication // SIAM News. 2005. November. Vol. 38, no. 9.

[3] Strassen V. Gaussian Elimination is not Optimal // Numerische Mathematik. 2005. Vol. 13, no. 9. P. 354–356.

[4] Погорелов Дмитрий Александрович Таразанов Артемий Михайлович Волкова Лилия Леонидовна. Оптимизация классического алгоритма Винограда для перемножения матриц // Журнал №1. 2019. Т. 49.

[5] Kotlin language specification [Электронный ресурс]. Режим доступа: `https://kotlinlang.org/spec/introduction.html` (дата обращения 09.10.2020.