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»

О Т Ч Е Т

по лабораторной работе № 0 2

Название ***Умножение матриц***

Дисциплина: ***Анализ алгоритмов***

Студент	***ИУ7И-56Б***		**Нгуен Ф. С.**
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			**Волокова Л. Л.**
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Оглавление

ВВЕДЕНИЕ 3

I. АНАЛИТИЧЕСКАЯ ЧАСТЬ 4

1. Описание алгоритмов 4
i. Классический алгоритм умножения 4
ii. Алгоритм Винограда 4
iii. Оптимизированный алгоритм Винограда 5

2. Модель вычислений 6

II. КОНСТРУКТОРСКАЯ ЧАСТЬ 7

1. Схемы алгоритмов 7

2. Оценка трудоёмкости 12
i. Классический алгоритм: 12
ii. Алгоритм Винограда: 12
iii. Оптимизированный алгоритм Винограда 13

III. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ 14

1. Требования к программному обеспечению: 14

2. Средства реализации 14

3. Реализации алгоритмов 14
i. Классический алгоритм 14
ii. Алгоритм Винограда 14
iii. Оптимизированный алгоритм Винограда 15

4. Тесты 16

IV. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ 17

1. Примеры работы 17

2. Сравнение работы алгоритмов при чётных размерах матрицы 17

3. Сравнение работы алгоритмов при нечётных размерах матрицы 18

ЗАКЛЮЧЕНИЕ 20

Введение

В огромном количестве областей научной и технической сферы деятельности человека при различных математических расчетах используют такую операцию как умножение матриц. Это довольно трудоемкий процесс даже при небольших размерах матриц, так как требуется большое количество операций умножения и сложения различных чисел. По этой причине человек озадачен проблемой оптимизации умножения матриц и ускорения процесса вычисления.

Таким образом, эффективное умножение матриц по времени и затратам ресурсов является актуальной проблемой для науки и техники.

Цель лабораторной работы - изучение трех алгоритмов умножения матриц: классического, алгоритма Винограда и его оптимизации.

Для того чтобы добиться этой цели, были поставлены следующие задачи:

- изучить и реализовать классический алгоритм умножения матриц и алгоритм Винограда
- оптимизировать работу алгоритма Винограда
- выполнить сравнительный анализ трудоёмкостей алгоритмов
- сравнить эффективность алгоритмов по времени

I. Аналитическая часть

1. Описание алгоритмов

i. Классический алгоритм умножения

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является таблицей, на пересечении строк и столбцов находятся элементы матрицы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две прямоугольные матрицы A и B размерности N * M, M * Q соответственно:

$$A(MxN) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{21} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix}$$

$$B(NxQ) = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1Q} \\ b_{21} & b_{21} & \cdots & b_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NQ} \end{bmatrix}$$

Тогда произведением матриц A и B называется матрица C размерностью M*Q:

$$C(MxQ) = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1Q} \\ c_{21} & c_{21} & \cdots & c_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M1} & c_{M2} & \dots & c_{MQ} \end{bmatrix}$$

$$c_{ij} = \sum_{k=0}^{N} a_{ik} b_{kj}, \qquad i = \overline{1, M}, j = \overline{1, Q}$$

ii. Алгоритм Винограда

Рассматривая результат умножения двух матриц очевидно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора:

$$\boldsymbol{U} = (\boldsymbol{u_1}, \boldsymbol{u_2}, \boldsymbol{u_3}, \boldsymbol{u_4})$$

$$\boldsymbol{V} = (\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \boldsymbol{v_4})$$

где $U = A_i$ *- i-ая строка матрицы A,*

$V = B_j$ *- j-ый столбец матрицы B*

Их скалярное произведение равно:

$$\boldsymbol{U} * \boldsymbol{V} = \boldsymbol{u_1 v_1} + \boldsymbol{u_2 v_1} + \boldsymbol{u_3 v_3} + \boldsymbol{u_4 v_4}$$

Это равенство можно переписать в виде:

$$\boldsymbol{U} * \boldsymbol{V} = (\boldsymbol{u_1} + \boldsymbol{v_1})(\boldsymbol{u_2} + \boldsymbol{v_2}) + (\boldsymbol{u_3} + \boldsymbol{v_3})(\boldsymbol{u_4} + \boldsymbol{v_4}) - \boldsymbol{u_1 u_2} - \boldsymbol{u_3 u_4} - \boldsymbol{v_1 v_2} - \boldsymbol{v_3 v_4}$$

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, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного.

iii. Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

- предварительные вычисления повторяющихся одинаковых действий
- уменьшения количества повторных проверок;
- замена цикл **for (int i = 0; i < N / 2; i++)** на **for (int i = 0; i< N – 1; i + =2)**

2. Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

- +, -, /, \%, ==, !=, <, >, <=, >=, [], ++, \-- -- имеют трудоемкость 1
- трудоемкость оператора выбора

  if (условие) then

  A

  Else

  B

  рассчитывается, как

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

- трудоемкость цикла рассчитывается, как

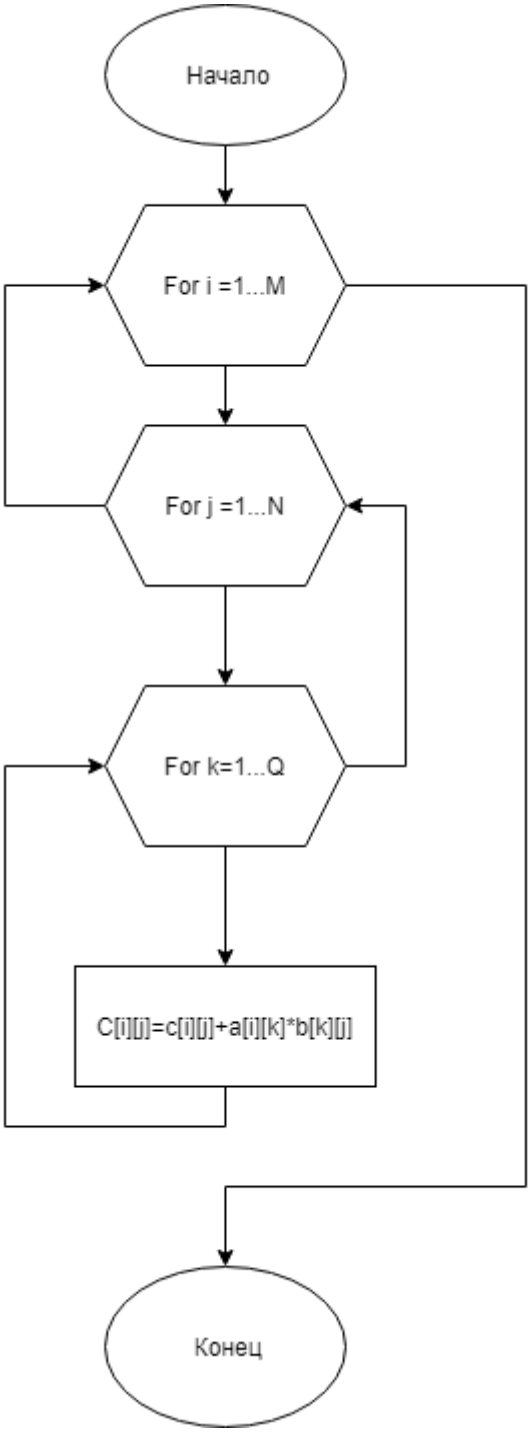
$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$$

- трудоемкость вызова метода равна 0
- трудоемкость вызова функции равна 1

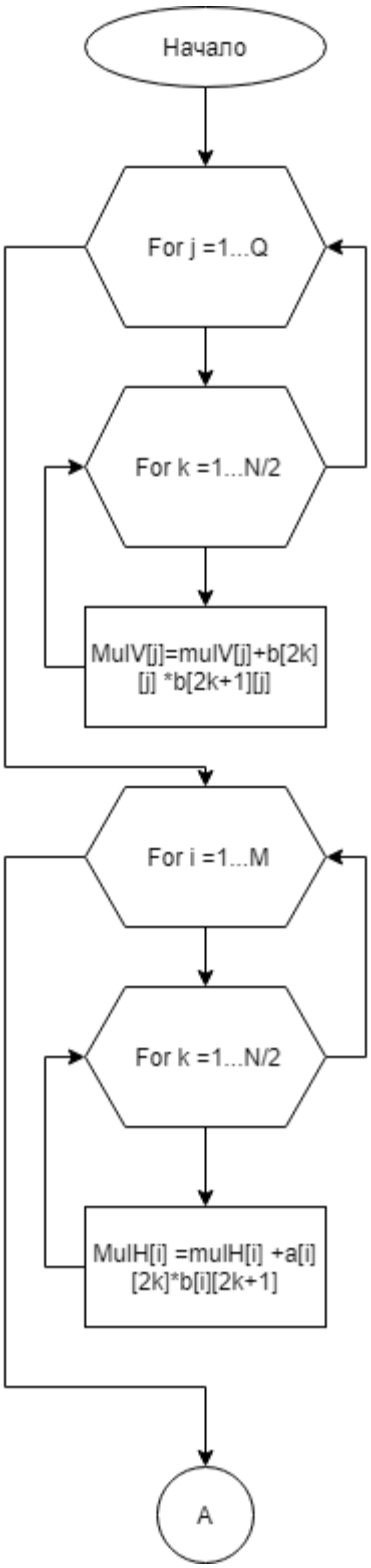
II. Конструкторская часть

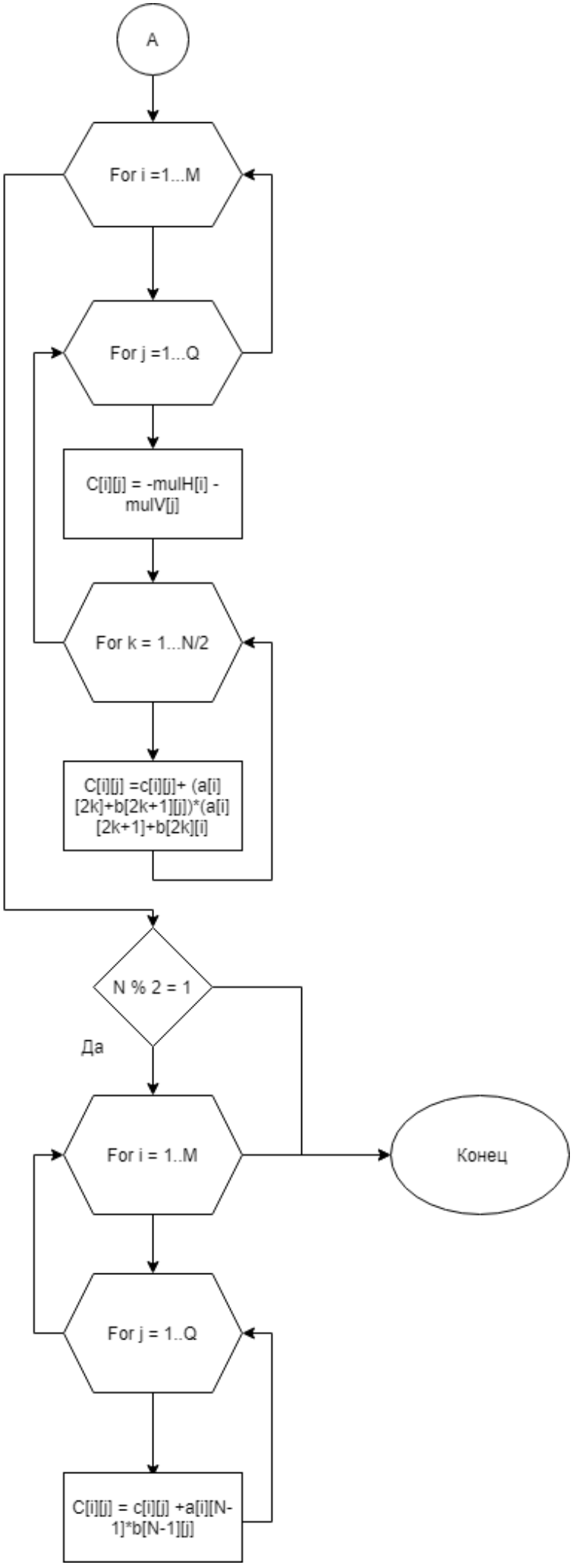
1. Схемы алгоритмов

Классический алгоритм

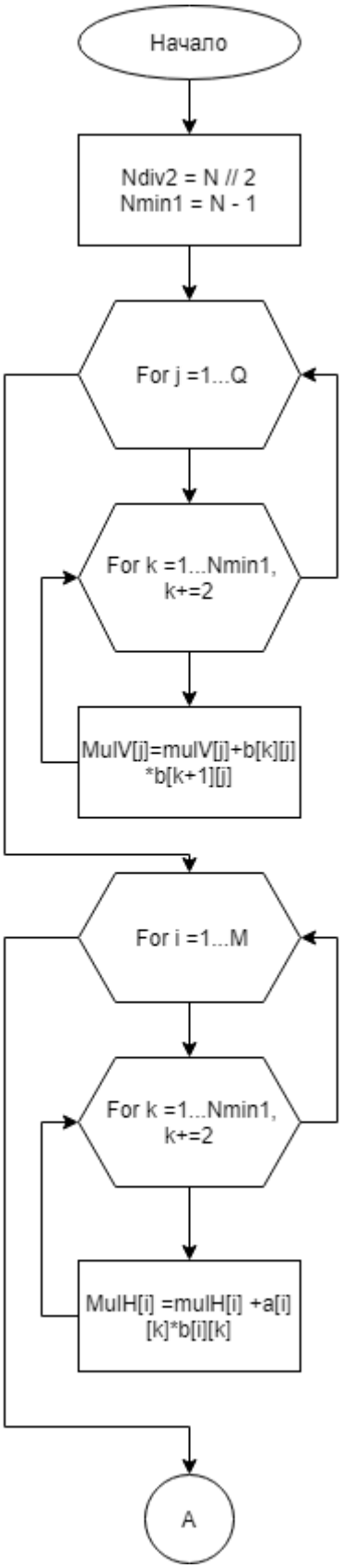


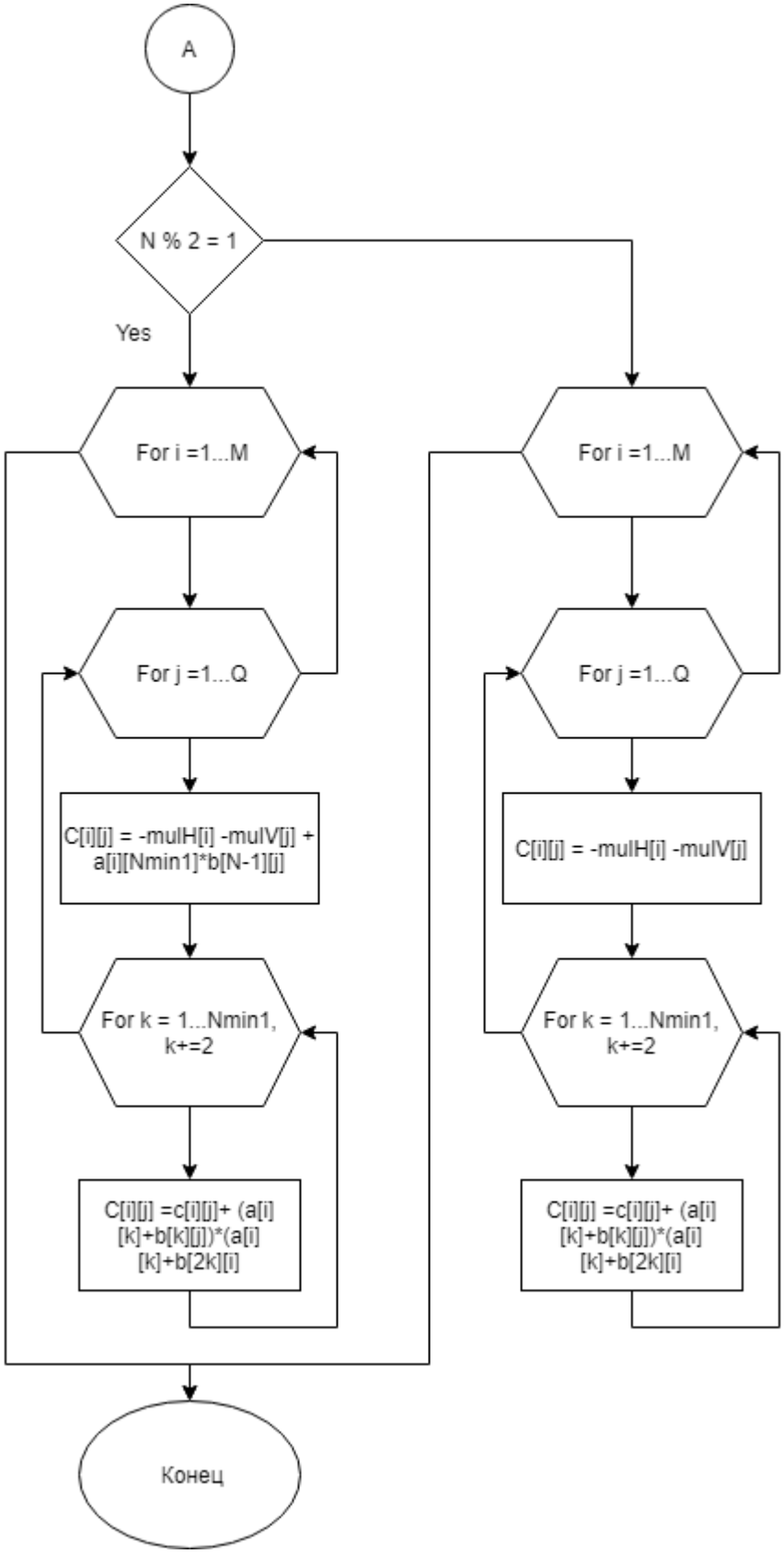
Алгоритм Винограда




A
For i =1...M
For j =1...Q
C[i][j] = -mulH[i] - mulV[j]
For k = 1...N/2
C[i][j] =c[i][j]+ (a[i][2k]+b[2k+1][j])*(a[i][2k+1]+b[2k][i]
N % 2 = 1
Да
For i = 1..M
Конец
For j = 1..Q
C[i][j] = c[i][j] +a[i][N-1]*b[N-1][j]

Оптимизированный алгоритм Винограда




A
N % 2 = 1
Yes
For i =1...M
For j =1...Q
C[i][j] = -mulH[i] -mulV[j] + a[i][Nmin1]*b[N-1][j]
For k = 1...Nmin1, k+=2
C[i][j] =c[i][j]+ (a[i][k]+b[k][j])*(a[i][k]+b[2k][i]
For i =1...M
For j =1...Q
C[i][j] = -mulH[i] -mulV[j]
For k = 1...Nmin1, k+=2
C[i][j] =c[i][j]+ (a[i][k]+b[k][j])*(a[i][k]+b[2k][i]
Конец

2. Оценка трудоёмкости

Пусть даны две матрицы A и B размерностью MxN и размерностью NxQ соответственно. Рассмотрим трудоемкость трёх алгоритмов умножения матриц.

i. *Классический алгоритм:*

$$f = 2 + M\left(2 + 2 + Q\big(2 + 2 + N(2 + 11)\big)\right) + 3 + 4M + 5MQ$$
$$= 13MNQ + 4MN + 5MQ + 8M + 5$$

Трудоемкость: O(MNQ)

ii. *Алгоритм Винограда:*

$$f_{\text{иниц}} = 3 + 4M + 5QM$$

$$f_I = \frac{15}{2}MN + AM + 2$$

$$f_{II} = \frac{15}{2}NQ + 4Q + 2$$

$$f_{III} = 2 + M(2 + 2 + Q(2 + 7 + 3 + \frac{N}{2}(3 + 1 + 12 + 5 + 5)$$
$$= 13MNQ + 12MQ + 4M + 2$$

$$f_{IV} = \begin{cases} 2, & \text{при чётном } N \\ 2 + 2 + M(2 + 2 + Q(2 + 13), & \text{при нечётном } N \\ = 15MQ + 4M + 4 \end{cases}$$

Общая трудоемкость алгоритма:

$$f = f_I + f_{II} + f_{III} + f_{IV} + f_{\text{иниц}}$$

$$= 13MNQ + 17MQ + \frac{15}{2}QN + \frac{15}{2}NM + 15M + 7Q + 9$$
$$+ \begin{cases} 2, & \text{при чётном } N \\ 15MQ + 4M + 4, & \text{при нечётном } N \end{cases}$$

Трудоемкость: O(MNQ)

iii. Оптимизированный алгоритм Винограда

$$f_{\text{иниц}} = 10 + 3 + 4M + 5QM$$

$$f_I = 2 + Q\left(2 + 2 + \frac{N}{2}(2 + 7)\right) = \frac{9}{2}NQ + 4Q + 2$$

$$f_{II} = 2 + M\left(2 + 2 + \frac{N}{2}(2 + 7)\right) = \frac{9}{2}MN + 4M + 2$$

$$f_{III} = 2 + M\left(2 + 2 + Q\left(2 + 7 + C + 2 + \frac{N}{2}(2 + 10 + 4 + 1 + 1)\right)\right)$$

$$= 9MNQ + 11MQ + 4M + 4 + \begin{cases} 0, \text{при чётном } N \\ 6MQ, \text{при нечётном } N \end{cases}$$

$$f = f_I + f_{II} + f_{III} + f_{\text{иниц}}{}^{+}$$

$$= 9MNQ + 16MQ + \frac{9}{2}NQ + \frac{9}{2}MN + 4Q + 12M + 15$$

$$\begin{cases} 0, \text{при чётном } N \\ 6MQ, \text{при нечётном } N \end{cases}$$

Трудоемкость: O(MNQ)

III. Технологическая часть

1. Требования к программному обеспечению:

На вход подаются размеры двух матриц. Матрицы генерируются случайным образом и выводятся на экран. На выход программа выдаёт три матрицы, которые являются результатами работы трёх различных алгоритмов умножения.

2. Средства реализации

Для реализации программы был использован язык Python. Для замера процессорного времени была использована функция time() из библиотеки time.

3. Реализации алгоритмов

i. *Классический алгоритм*

```
1.  def matrix_mult_stand(MatA, MatB):
2.      if MatA.col != MatB.row:
3.          return Mat()
4.
5.      M = MatA.row
6.      N = MatA.col
7.      Q = MatB.col
8.      MatC = matrix_create(M, Q)
9.
10.     for i in range(M):
11.         for j in range(Q):
12.             for k in range(N):
13.                 MatC.mat[i][j] = MatC.mat[i][j] + MatA.mat[i][k] * MatB.mat[k][j]
14.
15.     return MatC
```

ii. *Алгоритм Винограда*

```
1.  def matrix_mult_winograd(A, B):
2.      if A.col != B.row:
3.          return Mat()
4.
5.      M = A.row
6.      N = A.col
7.      Q = B.col
8.
9.      # I
10.     mulV = [0 for i in range(Q)]
11.     for j in range(Q):
12.         for k in range(N // 2):
13.             mulV[j] = mulV[j] + B.mat[2 * k][j] * B.mat[2 * k + 1][j]
14.
15.     #II
16.     mulH = [0 for i in range(M)]
17.     for i in range(M):
18.         for k in range(N // 2):
```

```
            mulH[i] = mulH[i] + A.mat[i][2 * k] * A.mat[i][2 * k + 1]

    #III
    C = matrix_create(M, Q)
    for i in range(M):
        for j in range(Q):
            C.mat[i][j] = -mulH[i] - mulV[j]
            for k in range(N // 2):
                C.mat[i][j] = C.mat[i][j] + (A.mat[i][2 * k] + B.mat[2 * k + 1][j])
 * (A.mat[i][2 * k + 1] + B.mat[2 * k][j])

    #IV
    if (N % 2 == 1):
        for i in range(M):
            for j in range(Q):
                C.mat[i][j] = C.mat[i][j] + A.mat[i][N - 1] * B.mat[N - 1][j]

    return C
```

iii. Оптимизированный алгоритм Винограда

```
def matrix_mult_5(A, B):

    if A.col != B.row:
        return Mat()
    M = A.row
    N = A.col
    Q = B.col
    Ndiv2 = N // 2
    Nmin1 = N - 1

    # I
    mulV = [0 for i in range(Q)]
    for j in range(Q):
        for k in range(0, Nmin1, 2):
            mulV[j] += B.mat[k][j] * B.mat[k + 1][j]

    #II
    mulH = [0 for i in range(M)]
    for i in range(M):
        for k in range(0, Nmin1, 2):
            mulH[i] += A.mat[i][k] * A.mat[i][k + 1]

    #III
    C = matrix_create(M, Q)

    if (N % 2 == 1):
        for i in range(M):
            for j in range(Q):
                C.mat[i][j] = -
mulH[i] - mulV[j] + A.mat[i][Nmin1] * B.mat[Nmin1][j]
                for k in range(0, Nmin1, 2):
                    C.mat[i][j] += (A.mat[i][k] + B.mat[k + 1][j]) * (A.mat[i][k +
1] + B.mat[k][j])


    else:
        for i in range(M):
            for j in range(Q):
                C.mat[i][j] = -mulH[i] - mulV[j]
                for k in range(0, Nmin1, 2):
                    C.mat[i][j] += (A.mat[i][k] + B.mat[k + 1][j]) * (A.mat[i][k +
1] + B.mat[k][j])


    return C
```

4. Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице:

A	B	C = AxB
[3]	[-5]	[-15]
$\begin{bmatrix} 4 & -1 & -1 \\ 0 & -2 & -4 \\ -1 & 1 & 4 \end{bmatrix}$	$\begin{bmatrix} 0 & 5 & -3 \\ 0 & -3 & 2 \\ 5 & -3 & 2 \end{bmatrix}$	$\begin{bmatrix} -5 & 26 & 16 \\ -20 & 18 & -12 \\ 20 & -20 & 13 \end{bmatrix}$
$\begin{bmatrix} 5 & 0 & -2 \end{bmatrix}$	$\begin{bmatrix} -3 & 0 \\ 5 & -4 \\ 1 & 2 \end{bmatrix}$	$\begin{bmatrix} -17 & -4 \end{bmatrix}$
$\begin{bmatrix} -1 \\ 4 \\ -4 \end{bmatrix}$	$\begin{bmatrix} 1 & -4 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 4 & 1 \\ 4 & -16 & -4 \\ -4 & 16 & 4 \end{bmatrix}$

В результате проверки реализации всех алгоритмов умножения прошли все поставленные функциональные тесты.

IV. Экспериментальная часть

1. Примеры работы

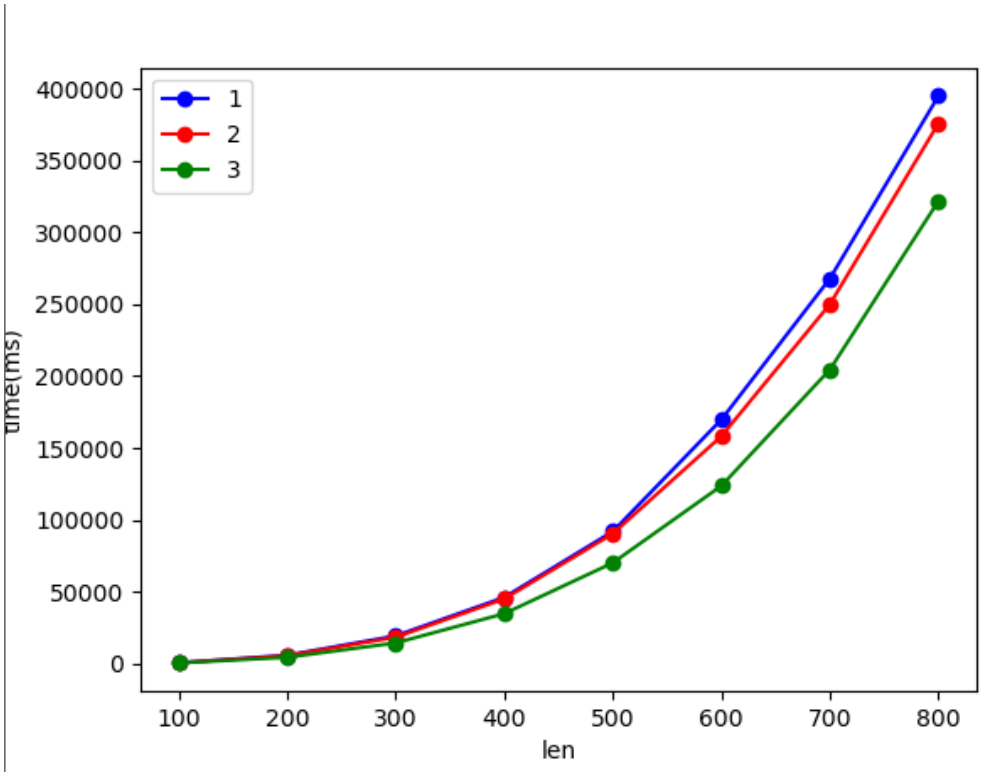
```
Matrix A:
2    4    -2    2    0    -1
0    4    -1    -1    4    -2
1    0    4    -3    -2    -1
-3    -1    -1    -3    0    -3
-3    3    4    5    0    5

Matrix B:
0    -1    -1    0
5    -2    3    0
0    -5    -4    -5
-3    -5    0    -3
4    4    2    -5
-1    -2    -2    3

Res:
15    -8    20    1
41    22    28    -18
2    -12    -19    -4
7    31    10    5
-5    -58    -14    -20
```

2. Сравнение работы алгоритмов при чётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 100 до 800 с шагом 100. Результаты измерений показаны в таблице и на рисунке.

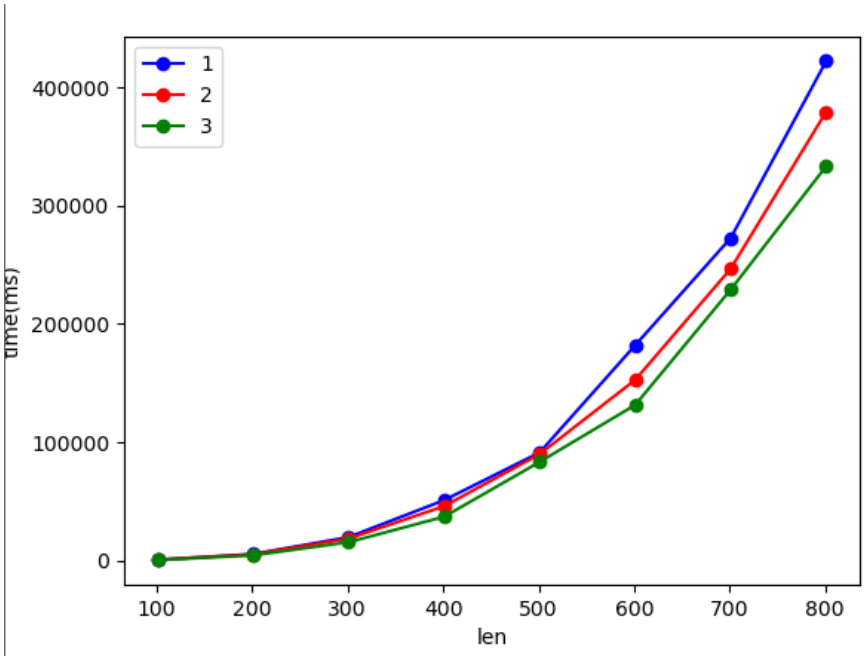


	1.Классический алгоритм	2.Алгоритм Винограда	3.Оптимизированный алгоритм Винограда
100	706	670	579
200	5942	5596	4472
300	19420	18470	14479
400	46200	45099	34869
500	92010	90497	70195
600	169787	158238	123730
700	267907	250059	204595
800	394933	375859	321825

Из результатов экспериментов можно сделать вывод о том, что алгоритм Винограда выигрывает классический алгоритм умножения. Оптимизированный алгоритм работает быстрее обычного алгоритма Винограда.

3. Сравнение работы алгоритмов при нечётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 101 до 801 с шагом 100. Результаты измерений показаны в таблице



	1.Классический алгоритм	2.Алгоритм Винограда	3.Оптимизированный алгоритм Винограда
101	710	699	582
201	5996	5612	4520
301	19882	18669	15639
401	50132	46080	38145
501	91433	90059	87468
601	182053	160794	131593
701	272717	253713	225085
801	412263	379459	333430

Для случая с нечётными размерами матриц можно сделать те же выводы, что и для случая с чётными. При этом можно заметить, что классический алгоритм в среднем работает за то же время, что и при чётных размерах, в то время как алгоритм Винограда и его оптимизация работают дольше за счёт дополнительных операций при нечётном случае. Однако по-прежнему классический алгоритм проигрывает по времени на те же величины.

Заключение

- ✓ Реализован классический алгоритм умножения матриц и алгоритм Винограда
- ✓ Оптимизированна работа алгоритма Винограда
- ✓ Выполнил сравнительный анализ трудоёмкостей алгоритмов
- ✓ Сравнил эффективность алгоритмов по времени

В ходе лабораторной работе были изучены и реализованы три алгоритма умножения матриц: классический алгоритм, алгоритм Винограда и его оптимизированный вариант. Сравнительный анализ алгоритмов показал, что алгоритмы Винограда, введя дополнительные векторы, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный и оптимизированный варианты работают быстрее классического алгоритма.