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 Алгоритмы умножения матриц

Студент Виноградов А. О.

Группа ИУ7-56Б

Оценка (баллы)

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

Москва — 2022 г.

Содержание

Введение

Матричная алгебра имеет обширные применения в различных отраслях знания – в математике, физике, информатике, экономике. Например, матрицы используется для решения систем алгебраических и дифференциальных уравнений, нахождения значений физических величин в квантовой теории, шифрования сообщений в Интернете.

Важной стороной работы с матрицами в программировании является оптимизация матричных операций (умножение, сложение, транспозиция и так далее), так как во многих задачах размеры матриц могут достигать больших значений. В данной лабораторной работе пойдет речь об оптимизации операции умножения матриц.

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов умножения матриц — классический алгоритм, алгоритм Винограда, оптимизированный алгоритм Винограда.

Задачи данной лабораторной работы:

1) изучение и реализация алгоритмов умножения матриц — классического, Винограда и его оптимизацию;

2) проведение сравнительного анализа по времени работы классического алгоритма и алгоритма Винограда;

3) проведение сравнительного анализа по времени работы алгоритма Винограда и его оптимизации;

4) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе разобраны классический алгоритм умножения матриц, алгоритм Винограда, а также его оптимизация.

1.1 Матрица

Матрицей [1] называется прямоугольная таблица чисел, содержащая m строк одинаковой длины (или n столбцов одинаковой длины). Матрица записывается в виде:

$$A_{m\times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \tag{1.1}$$

Числа a_{ij}, составляющие матрицу, называются ее элементами.

Можно выделить следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) умножение матрицы на число;

3) умножение матрицы на матрицу.

Умножение матрицы на матрицу возможно, если количество столбцов первой матрицы равно количеству строк второй матрицы. В итоговой матрице количество строк будет, как у первой матрицы, а столбцов — как у второй.

1.2 Классический алгоритм

Пусть даны две матрицы:

$$A_{m\times p} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{pmatrix} B_{p\times n} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pn} \end{pmatrix}. \quad (1.2)$$

Произведением матрицы A на матрицу B называется матрица:

$$C_{m\times n} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix}, \quad (1.3)$$

где

$$c_{ij} = \sum_{r=1}^{p} a_{ir} b_{rj} \quad (i = \overline{1,m}; j = \overline{1,n}) \quad (1.4)$$

Классический алгоритм реализует формулу (1.3).

1.3 Алгоритм Винограда

Алгоритм Винограда [2] — алгоритм умножения матриц, основанный на предвычислении значений для ускорения операции матричного умножения.

Рассмотрим два вектора $U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)$.
Скалярное произведение $U \cdot V^T$ равно:

$$U \cdot V^T = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4. \quad (1.5)$$

Это уравнение эквивалентно:

$$U \cdot V^T = (u_1+v_2)(u_2+v_1)+(u_3+v_4)(u_4+v_3)-u_1u_2-u_3u_4-v_1v_2-v_3v_4. \quad (1.6)$$

Произведения простых сомножителей (последние четыре в правой части равенства) можно вычислить заранее и запомнить для каждой строки первой

матрицы и каждого столбца второй матрицы. За счет этого можно добиться прироста производительности: для вычисления одного элемента итоговой матрицы (произведения) потребуется выполнить два умножения и пять сложений, а затем прибавить к результату заранее запомненные для данной строки первой матрицы и столбца второй матрицы значения. Операция сложения выполняется быстрее операции умножения, следовательно, на практике данный алгоритм должен работать быстрее.

Следует учитывать, что в случае нечетного размера матрицы требуется дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Оптимизация алгоритма Винограда

В программной реализации описанного выше алгоритма Винограда можно использовать следующие оптимизации:

1) заменить операцию $x = x + k$ на $x+ = k$;

2) заменить умножение на 2 на побитовый сдвиг;

3) предвычислять некоторые слагаемые из алгоритма.

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц — классический, алгоритм Винограда и его оптимизация.

2 Конструкторская часть

В данном разделе разработаны схемы исследуемых алгоритмов, оценены трудоемкости в лучших и худших случаях.

2.1 Разработка алгоритмов

В данной части будут рассмотрены схемы классического алгоритма умножения матриц, алгоритма Винограда и его оптимизации. На рисунках 2.1 – 2.5 представлены схемы рассматриваемых алгоритмов.

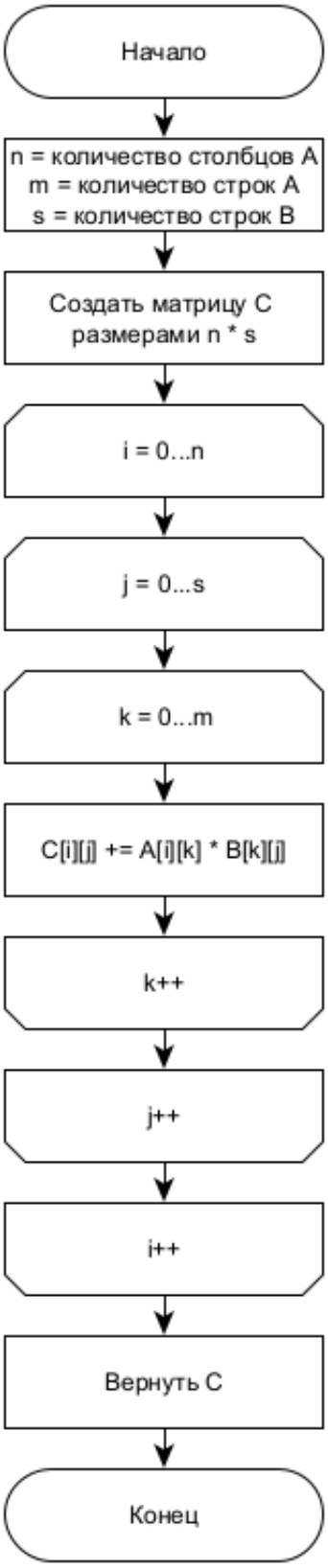

Рисунок 2.1 – Схема классического алгоритма умножения матриц

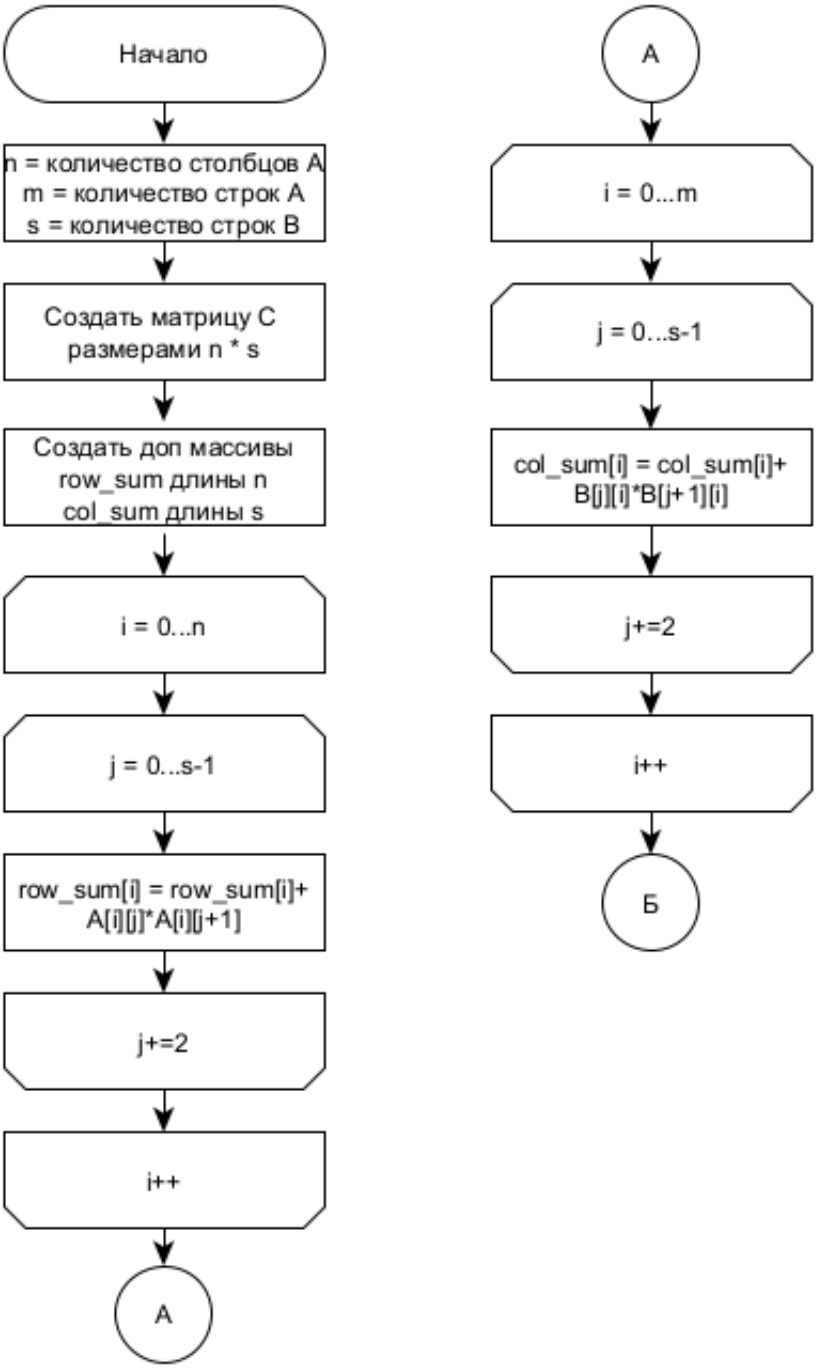

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц

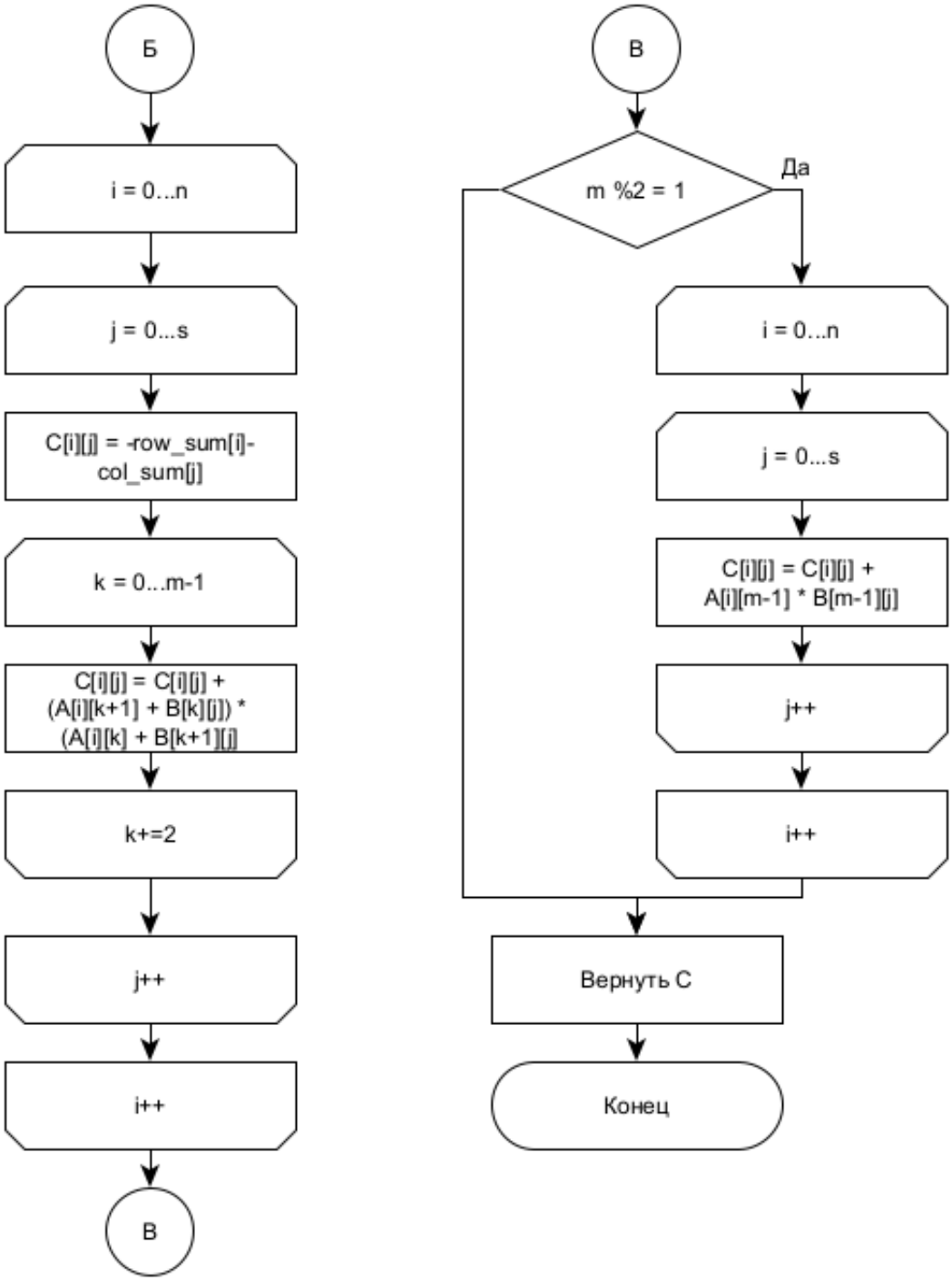

Рисунок 2.3 – Схема алгоритма Винограда умножения матриц

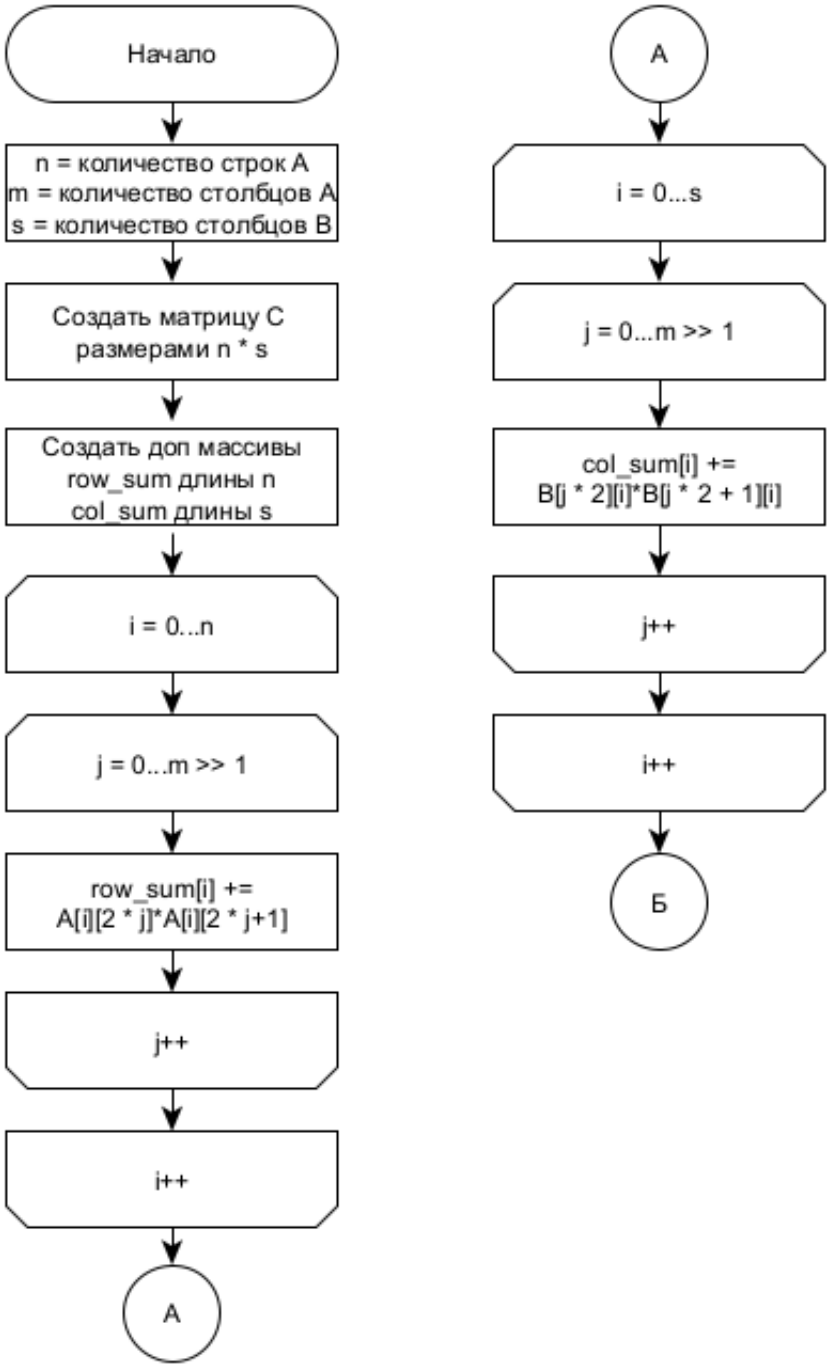

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда умножения матриц

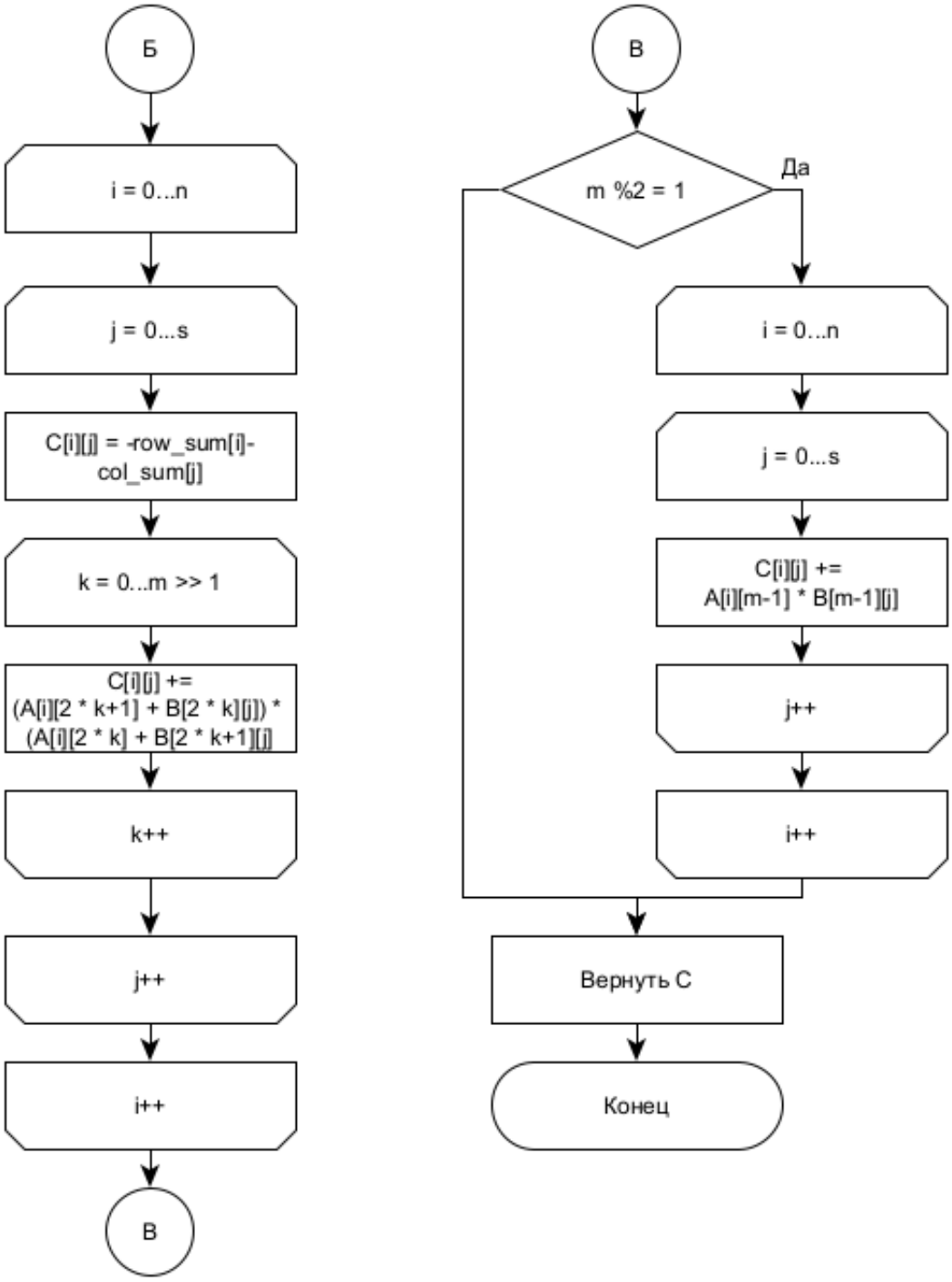

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда умножения матриц

2.2 Модель вычислений

Для последующего вычисления трудоемкости введем модель вычислений [3]:

1) Операции из списка (2.1) имеют трудоемкость 1.

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

2) Трудоемкость оператора выбора *if 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 (2.3).

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

4) Трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В этой части рассчитаны трудоемкости рассматриваемых алгоритмов.

2.3.1 Классический алгоритм

Трудоемкость классического алгоритма умножения матриц состоит из следующих элементов:

– трудоемкость внешнего цикла $i \in [0...n)$:

$$f = 2 + n \cdot (2 + f_{\text{ц1}}); \tag{2.4}$$

– трудоемкость цикла $j \in [0...s)$:

$$f_{\text{ц1}} = 2 + s \cdot (2 + f_{\text{ц2}}); \tag{2.5}$$

– трудоемкость цикла $k \in [0...m)$:

$$f_{\text{ц2}} = 2 + 13m. \tag{2.6}$$

Суммарная трудоемкость классического алгоритма умножения матриц:

$$f_{\text{класс}} = 2 + n(4 + s(4 + 13m)) = 13nms + 4ns + 4n + 2 \approx 13nms \tag{2.7}$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда умножения матриц состоит из следующих элементов:

– трудоемкость первого двойного цикла $j \in [0...\frac{m}{2})$ для $i \in [0...n)$:

$$f_1 = 2 + n(3 + \frac{m}{2}(3 + 12)); \tag{2.8}$$

– трудоемкость второго двойного цикла $j \in [0...\frac{m}{2})$ для $i \in [0...s)$:

$$f_2 = 2 + s(3 + \frac{m}{2}(3 + 12)); \tag{2.9}$$

– трудоемкость тройного цикла $k \in [0...\frac{m}{2})$ для $j \in [0...s)$ для $i \in [0...n)$:

$$f_3 = 2 + n(2 + s(2 + 7 + 3 + \frac{m}{2}(3 + 23))); \quad (2.10)$$

– трудоемкость оператора выбора с двойным циклом $j \in [0...s)$ для $i \in [0...n)$ (в худшем случае):

$$f_{\text{усл}} = 2 + \begin{cases} 0, & \text{в лучшем случае} \\ 2 + n(2 + s(2 + 13)), & \text{в худшем случае} \end{cases}. \quad (2.11)$$

Суммарная трудоемкость алгоритма Винограда умножения матриц в лучшем случае:

$$f_{\text{вин\_лудш}} = 13nms + 12ns + \frac{15}{2}mn + \frac{15}{2}ms + 5n + 3s + 8 \approx 13nms \quad (2.12)$$

Суммарная трудоемкость алгоритма Винограда умножения матриц в худшем случае:

$$f_{\text{вин\_худш}} = 13nms + 27ns + \frac{15}{2}mn + \frac{15}{2}ms + 5n + 3s + 10 + 2n \approx 13nms \quad (2.13)$$

2.3.3 Оптимизированный алгоритм Винограда

В ходе оценки трудоемкости используются следующие обозначения:

– $m_{05} = \frac{m}{2}$

– $j_{05} = \frac{j}{2}$

– $j_{051} = j_{05} + 1$

Трудоемкость оптимизированного алгоритма Винограда умножения матриц состоит из следующих элементов:

– трудоемкость первого двойного цикла $j \in [0...m_05)$ для $i \in [0...n)$:

$$f_1 = 2 + n(2 + \frac{m}{2}(2 + 10)); \quad (2.14)$$

– трудоемкость второго двойного цикла $j \in [0...m_05)$ для $i \in [0...s)$:

$$f_2 = 2 + s(3 + \frac{m}{2}(2 + 10)); \quad (2.15)$$

– трудоемкость тройного цикла $k \in [0...m_05)$ для $j \in [0...s)$ для $i \in [0...n)$:

$$f_3 = 2 + n(2 + s(2 + 7 + 3 + \frac{m}{2}(2 + 18))); \tag{2.16}$$

– трудоемкость оператора выбора с двойным циклом $j \in [0...s)$ для $i \in [0...n)$ (в худшем случае):

$$f_{\text{усл}} = 2 + \begin{cases} 0, & \text{в лучшем случае} \\ 2 + n(2 + s(2 + 10)), & \text{в худшем случае} \end{cases}. \tag{2.17}$$

Суммарная трудоемкость оптимизированного алгоритма Винограда умножения матриц в лучшем случае:

$$f_{\text{опт\_лучш}} = 10nms + 12ns + 6mn + 6ms + 5n + 3s + 8 \approx 10nms \tag{2.18}$$

Суммарная трудоемкость оптимизированного алгоритма Винограда умножения матриц в худшем случае:

$$f_{\text{опт\_худш}} = 10nms+12ns+6mn+6ms+5n+3s+10+2n+12ns \approx 10nms \tag{2.19}$$

Вывод

На основе теоретических данных, полученных в аналитическом разделе были построены схемы исследуемых алгоритмов, посчитаны теоретические трудоемкости алгоритмов в лучших и худших случаях.

3 Технологическая часть

В данном разделе рассмотрены средства реализации, а также представлены листинги реализаций алгоритмов умножения матриц — классического, алгоритма Винограда и его оптимизации.

3.1 Средства реализации

Для данной работы был выбран язык Python [4]. Для данной лабораторной работы требуются инструменты для работы с массивами, замеров процессорного времени работы выполняемой программы, визуализации полученных данных. Все перечисленные инструменты присутствуют в выбранном языке программирования

3.2 Разработка алгоритмов

В листингах 3.1 — 3.3 приведены реализации алгоритмов умножения матриц.

Листинг 3.1 – Функция умножения матриц по классическому алгоритму

```
1  def standart(m1: [[int]], m2:[[int]]) ->[[int]] :
2      if  len(m1[0]) != len(m2):
3          return [[]]
4      n = len(m1)
5      m = len(m1[0])
6      s = len(m2[0])
7      res = [[0] * s for i in range(n)]
8      for i in range(n):
9          for k in range(s):
10             for j in range(m):
11                 res[i][k] = res[i][k] + m1[i][j] * m2[j][k]
12     return res
```

Листинг 3.2 – Функция умножения матриц по алгоритму Винограда

```
1
2       n = len(m1)
3       m = len(m1[0])
4       s = len(m2[0])
5
6       res = [[0] * s for i in range(n)]
7       row_sum = [0] * n
8       col_sum = [0] * s
9
10      for i in range(n):
11          for j in range(m//2):
12              row_sum[i] = row_sum[i] + m1[i][2 * j] * m1[i][2 * j +
                    1]
13
14      for i in range(s):
15          for j in range(m//2):
16              col_sum[i] = col_sum[i] + m2[2 * j][i] * m2[2 * j +
                    1][i]
17
18      for i in range(n):
19          for j in range(s):
20              res[i][j] = -row_sum[i] - col_sum[j]
21              for k in range(0, m // 2):
22                  res[i][j] = res[i][j] + (m1[i][2 * k + 1] + m2[2 *
                        k][j]) * (m1[i][2 * k] + m2[2 * k + 1][j])
23
24      if (m % 2 == 1):
25          for i in range(n):
26              for j in range(s):
27                  res[i][j] = res[i][j] + m1[i][m - 1] * m2[m - 1][j]
28
29      return res
```

Листинг 3.3 – Функция умножения матриц по оптимизированному алгоритму Винограда

```
def win_alg_opt(m1: [[int]], m2:[[int]]) ->[[int]] :
    if  len(m1[0]) != len(m2):
        print("Error: sizes doesn't match - multiplication
           impossible")
        return [[]]

    n = len(m1)
    m = len(m1[0])
    s = len(m2[0])

    res = [[0] * s for i in range(n)]
    row_sum = [0] * n
    col_sum = [0] * s

    m_05 = m >> 1
    for i in range(n):
        for j in range(m_05):
            t = j << 1
            row_sum[i] += m1[i][t] * m1[i][t + 1]

    for i in range(s):
        for j in range(m_05):
            t = j << 1
            col_sum[i] += m2[t][i] * m2[t + 1][i]

    for i in range(n):
        for j in range(s):
            res[i][j] = -row_sum[i] - col_sum[j]
            for k in range(0, m_05):
                t = k << 1
                t1 = t + 1
                res[i][j] += (m1[i][t1] + m2[t][j]) * (m1[i][t] +
                   m2[t1][j])
    if (m % 2 == 1):
        for i in range(n):
            for j in range(s):
                t = m - 1
                res[i][j] += m1[i][t] * m2[t][j]

    return res
```

3.3 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы умножения матриц.

Таблица 3.1 – Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 7 \end{pmatrix}$	$\begin{pmatrix} 8 \end{pmatrix}$	$\begin{pmatrix} 56 \end{pmatrix}$

Все тесты пройдены функциями успешно.

Вывод

В данном разделе были разработаны и приведены исходные коды трех алгоритмов, рассмотренных и описанных ранее. Также, были описаны выбранные средства реализации алгоритмов и функциональные тесты для описанных алгоритмов.

4 Исследовательская часть

Демонстрация работы программы приведена на рисунке (4.1).

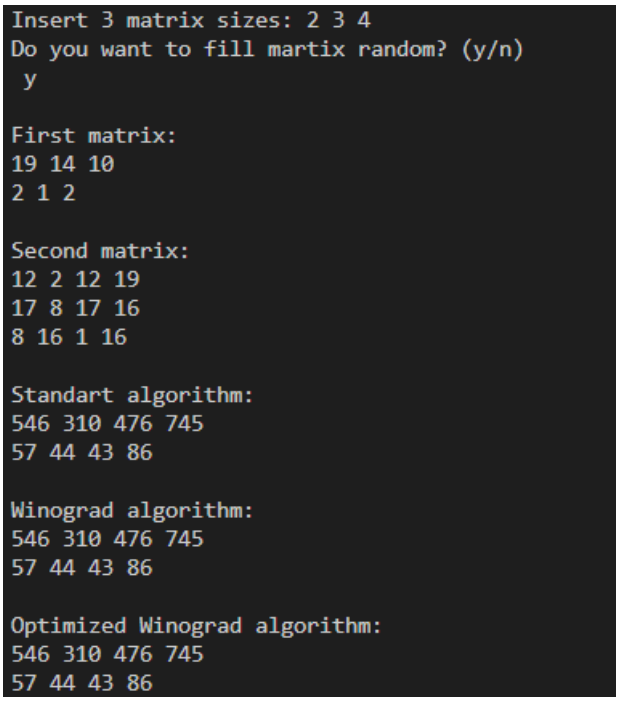

Рисунок 4.1 – Пример работы программы для вычисления произведения матриц тремя алгоритмами

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено измерение времени работы ПО:

– операционная система Windows 10 Домашняя Версия 21H1 [5] x86_64;

– оперативная память 8 Гб 2133 МГц;

– процессор Intel Core i5-8300H с тактовой частотой 2.30 ГГц [6].

4.2 Время выполнения реализаций алгоритмов

Для замеров времени использовалась функция замера процессорного времени process_time из библиотеки time на Python. Функция возвращает процессорное время типа float [7].

Функция используется дважды — в начале и в конце замера времени, затем значения начальное значение вычитается из конечного.

Замеры проводились для квадратных матриц размерами 200, 300,...,1000 для четной размерности и размерами 201, 301,...,1001 для нечетной размерности. Матрицы заполнялись случайными числами.

В таблицах 4.1 – 4.2 представлены замеры времени работы для каждого из алгоритмов.

Таблица 4.1 – Результаты замеров времени работы реализаций алгоритмов на матрицах четной размерности (в мс)

Размерность	Классический алг.	Алг. Винограда	Опт. алг. Винограда
200	890.625	921.875	875.0
300	3078.125	3250.0	2937.5
400	7234.375	8031.25	6984.375
500	14 328.125	16 156.25	13 640.625
600	26 046.875	28 906.25	24 406.25
700	40 250.0	47 265.625	38 875.0
800	60 750.0	71 312.5	58 468.75
900	93 343.75	104 062.5	84 250.0
1000	123 406.25	145 187.5	116 578.125

Таблица 4.2 – Результаты замеров времени работы реализаций алгоритмов на матрицах нечетной размерности (в мс)

Размерность	Классический алг.	Алг. Винограда	Опт. алг. Винограда
201	906.25	921.875	890.625
301	3062.5	3265.625	2968.75
401	7343.75	8140.625	7125.0
501	15 140.625	16 234.375	13 781.25
601	25 078.125	28 890.625	24 484.375
701	40 687.5	47 250.0	39 140.625
801	64 093.75	72 906.25	59 031.25
901	89 312.5	104 296.875	84 515.625
1001	125 156.25	146 218.75	118 000.0

На рисунках 4.2 – 4.3 приведены графические результаты замеров.

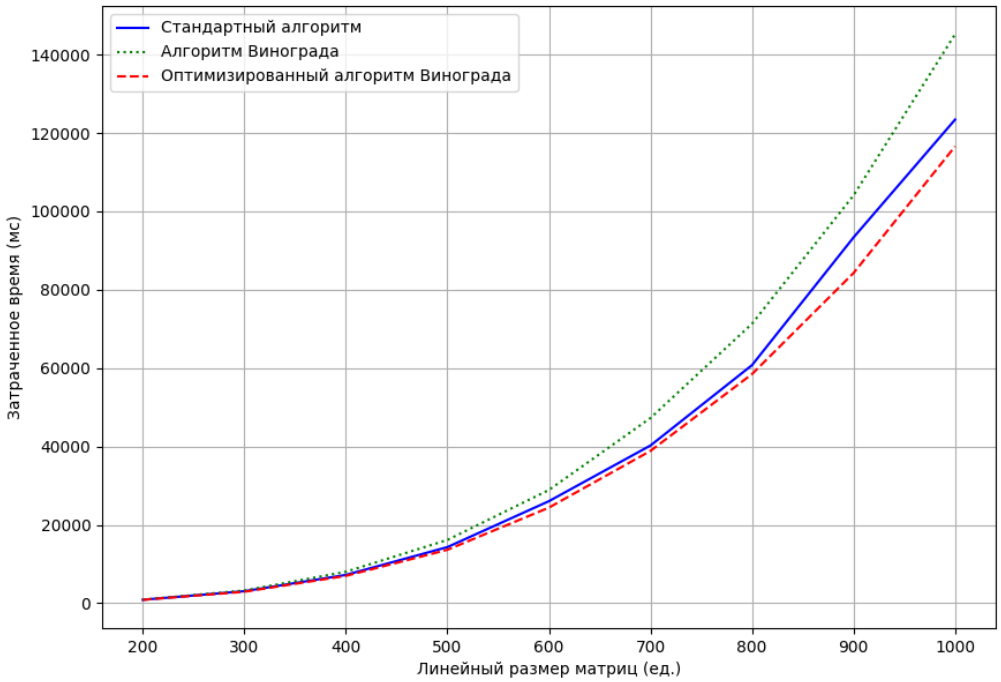

Рисунок 4.2 – Результаты замеров времени работы реализаций алгоритмов на матрицах четной размерности

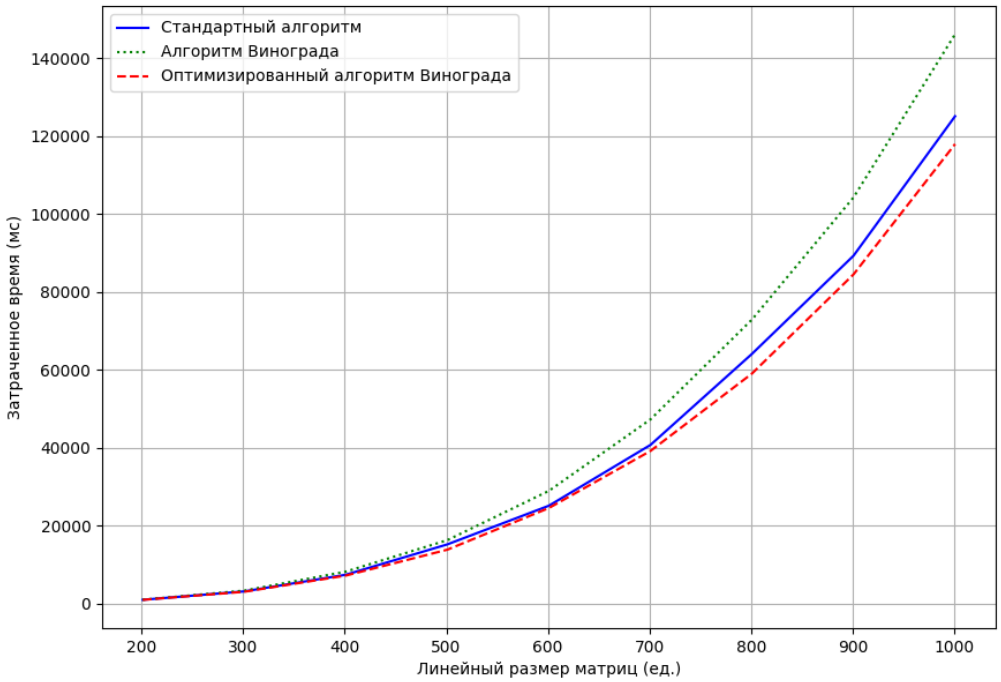

Рисунок 4.3 – Результаты замеров времени работы реализаций алгоритмов на матрицах нечетной размерности

Из полученных результатов можно сделать вывод, что классический алгоритм работает быстрее алгоритма Винограда (на четных и нечетных размерностях матриц), однако уступает оптимизированному алгоритму Винограда.

Вывод

В результате экспериментов было выявлено, что реализация классического алгоритма умножения матриц работает быстрее реализации алгоритма Винограда (на четных и нечетных размерностях матриц), однако уступает оптимизированной реализации алгоритма Винограда. Для квадратных матриц размерностью 1000×1000 реализации дают следующие результаты:

- классический алгоритм — 123,4 с;
- алгоритм Винограда — 145,2 с;
- оптимизированный алгоритм Винограда — 118,0 с.

Таким образом, использование оптимизированного алгоритма Винограда является более приоритетным при необходимости ускорить выполнение произведения матриц. Однако следует учитывать, что данный алгоритм влечет увеличение затрат памяти для хранение дополнительных массивов.

Заключение

В ходе выполнения лабораторной работы было выявлено, что реализация классического алгоритма выполняет быстрее реализации алгоритма Винограда, но медленнее оптимизированного алгоритма Винограда. Для квадратных матриц размерностью 1000×1000 реализации дают следующие результаты:

- классический алгоритм — 123,4 с;
- алгоритм Винограда — 145,2 с;
- оптимизированный алгоритм Винограда — 118,0 с.

Таким образом, оптимизированный алгоритм Винограда умножения матриц работает с квадратными матрицами размерностью 1000×1000 быстрее классического алгоритма в 1,07 раз и быстрее алгоритма Винограда (неоптимизированного) в 1,24 раз.

Цель, поставленная в начале работы, была достигнута. Кроме того были достигнуты поставленные задачи:

- были изучены и реализованы алгоритмы умножения матриц — классического, Винограда и его оптимизация;
- был проведен сравнительный анализ по времени работы классического алгоритма и алгоритма Винограда;
- был проведен сравнительный анализ по времени работы алгоритма Винограда и его оптимизации;
- был подготовлен отчет о лабораторной работе.

Список использованных источников

[1] Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. – М.: Гуманит. изд. центр ВЛАДОС, 2003. — С. 45–49.

[2] Kakaradov B. Ultra-Fast Matrix Multiplication: An Empirical Analysis of Highly Optimized Vector Algorithms [Электронный ресурс] // cs.stanford.edu: [сайт]. [2004]. URL: https://cs.stanford.edu/people/boyko/pubs/MatrixMult_SURJ_2004.pdf.

[3] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ.

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 10.10.2022).

[5] Windows [Электронный ресурс]. Режим доступа: https://www.microsoft.com/en-us/windows (дата обращения: 01.10.2022).

[6] Процессор Intel Core i5 [Электронный ресурс]. Режим доступа: https://www.intel.com/processors/core/i5/docs (дата обращения: 01.10.2022).

[7] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 10.10.2022).