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»

Лабораторная работа №2

По предмету: «Анализ алгоритмов»

Алгоритм Винограда

Студент: Гасанзаде М.А.,
Группа: ИУ7-56Б

Москва, 2019 г.

Оглавление

Введение

Две матрицы можно перемножить, если число столбцов в первой матрице совпадает с числом строк во второй. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Для вычисления произведения двух матриц каждая строка первой почленно умножается на каждый столбец второй. Затем подсчитывается сумма таких произведений и записывается в соответствующую клетку результата.

Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга.

Цель работы: изучение метода динамического программирования на материале алгоритма Винограда и наборе оптимизаций.

Задачи работы:

1) изучение алгоритма Винограда и оптимизаций для умножения матриц;

2) применение метода динамического программирования для реализации алгоритма Винограда;

3) получение практических навыков реализации указанных алгоритмов: стандартного алгоритма, алгоритма Винограда и алгоритма Винограда с набором оптимизаций;

4) сравнительный анализ реализаций алгоритмов умножения матриц по затрачиваемым ресурсам (времени и памяти);

5) экспериментальное подтверждение различий во временно́й эффективности реализаций алгоритмов умножения матриц при помощи

разработанного программного обеспечения на материале замеров процессорного времени выполнения реализаций на варьирующихся размерах матриц;

6) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1. Аналитическая часть

В данной части дано теоретическое описание алгоритмов и указание области их применения.

1.1. Описание алгоритмов

Пусть A и B — матрицы, произведение которых C требуется найти; M * N и N * Q — их размер соответственно. Следующий псевдокод иллюстрирует стандартный алгоритм умножения матриц:

***Рисунок* 1. Стандартный алгоритм умножения матриц**

```
for i = 1 to M do
    for j = 1 to Q do
        for k = 1 to N do
            C[i][j] = C[i][j] + A[i][k] * B[k][j]
        end for k
    end for j
end for i
```

Можно заметить, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. Это реализовано в алгоритме Винограда[5].

Рассмотрим два вектора: $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно: $v1w1 + v2w2 + v3w3 + v4w4$.

Это равенство можно переписать в виде: $(v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$.

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.

Вычисление результирующей матрицы (*формула 1*)

$$C_{i,j} = -Rows_i - Cols_j + \sum_{k=1}^{n/2} (A_{i,2k+1} + B_{2k,j}) \cdot (A_{i,2k} + B_{2k+1,j})$$

Алгоритм Винограда можно оптимизировать. Если объединить последние два цикла, то можно сэкономить ресурсы на их обслуживании. Следует также накапливать значение очередного элемента результирующей матрицы в буфер, что позволит выполнять меньше адресаций. А вычислив заранее значение N – 1, можно в дальнейшем использовать его без лишних вычислительных трат. Будем учитывать, что лучший случай — это матрица с чётной размерностью, и худший случай — когда количество столбцов в матрице A или количество строк в матрице B нечётное.

При худшем случае к результирующей формуле(*формула 1*) будет применено следующее выражение: $C_{i,j} += A_{i,n} * B_{n,j}$ Где A&B это матрицы размерами [m*n]&[n*k] соответственно, C это вычисляемая матрица с размерами [m*k]

1.2. Применение алгоритмов

Алгоритмы активно применяются в 2, 3 пунктах:

1. расчётах трёхмерной графики;
2. математическом анализе при интегрировании систем дифференциальных уравнений, в теории вероятностей[2];
3. исследовании линейных отображений векторных пространств, линейных и квадратичных форм, систем линейных уравнений;

4. экономике для обработки балансово-нормативных моделей, отражающих соотношения затрат и результатов производства, нормативы затрат, производственные и экономические структуры.

2. Конструкторская часть

В данной части приведены схемы алгоритмов, а также их сравнительный анализ.

2.1 Разработка алгоритмов

Функциональная модель процесса умножения матриц в нотации IDEF0 представлена на рис. 1.

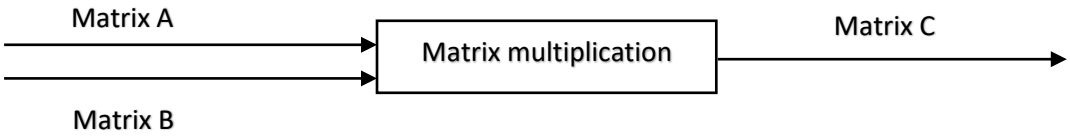
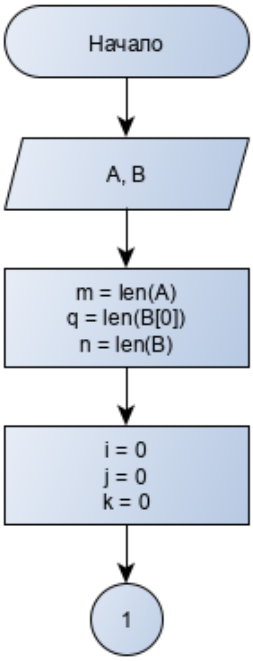


Рисунок 1 – Функциональная модель процесса умножения матриц

Схемы алгоритмов представлены на *рис*. 3, 4, 5.



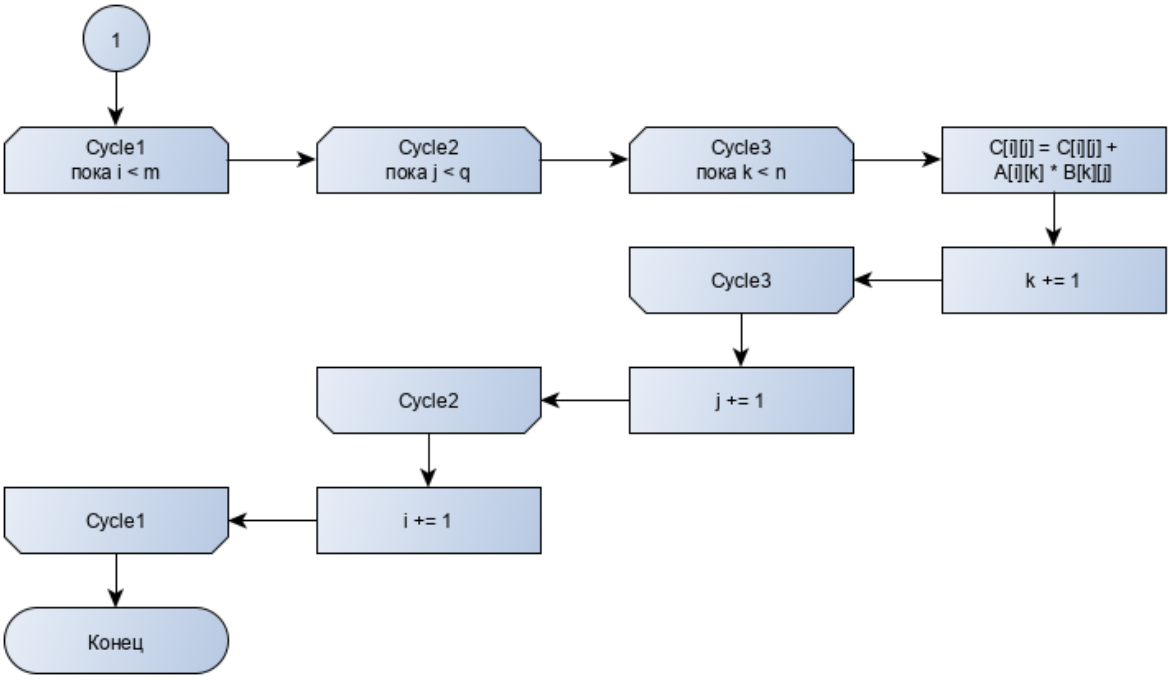
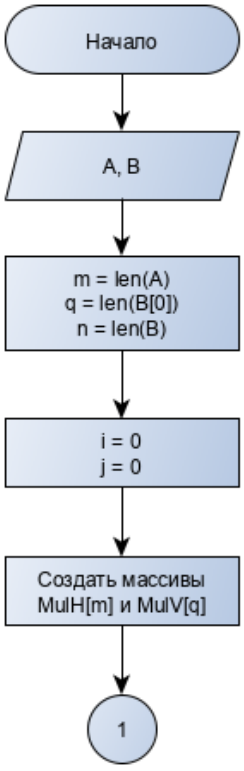
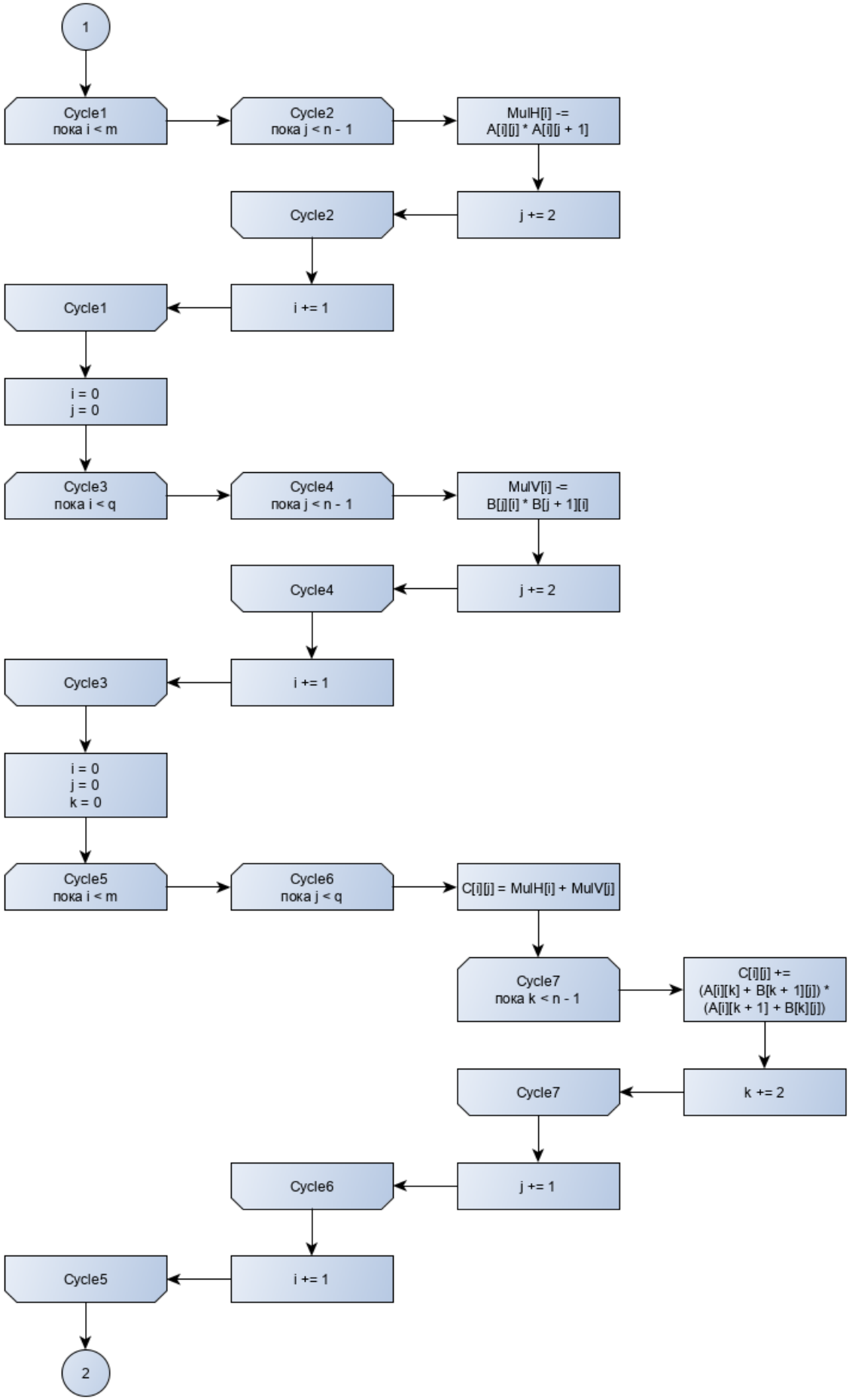


Рисунок 3 – Схема стандартного алгоритма умножения матриц




1
Cycle1
пока i < m
Cycle2
пока j < n - 1
MulH[i] -=
A[i][j] * A[i][j + 1]
j += 2
Cycle2
i += 1
Cycle1
i = 0
j = 0
Cycle3
пока i < q
Cycle4
пока j < n - 1
MulV[i] -=
B[j][i] * B[j + 1][i]
j += 2
Cycle4
i += 1
Cycle3
i = 0
j = 0
k = 0
Cycle5
пока i < m
Cycle6
пока j < q
C[i][j] = MulH[i] + MulV[j]
Cycle7
пока k < n - 1
C[i][j] +=
(A[i][k] + B[k + 1][j]) *
(A[i][k + 1] + B[k][j])
k += 2
Cycle7
j += 1
Cycle6
i += 1
Cycle5
2

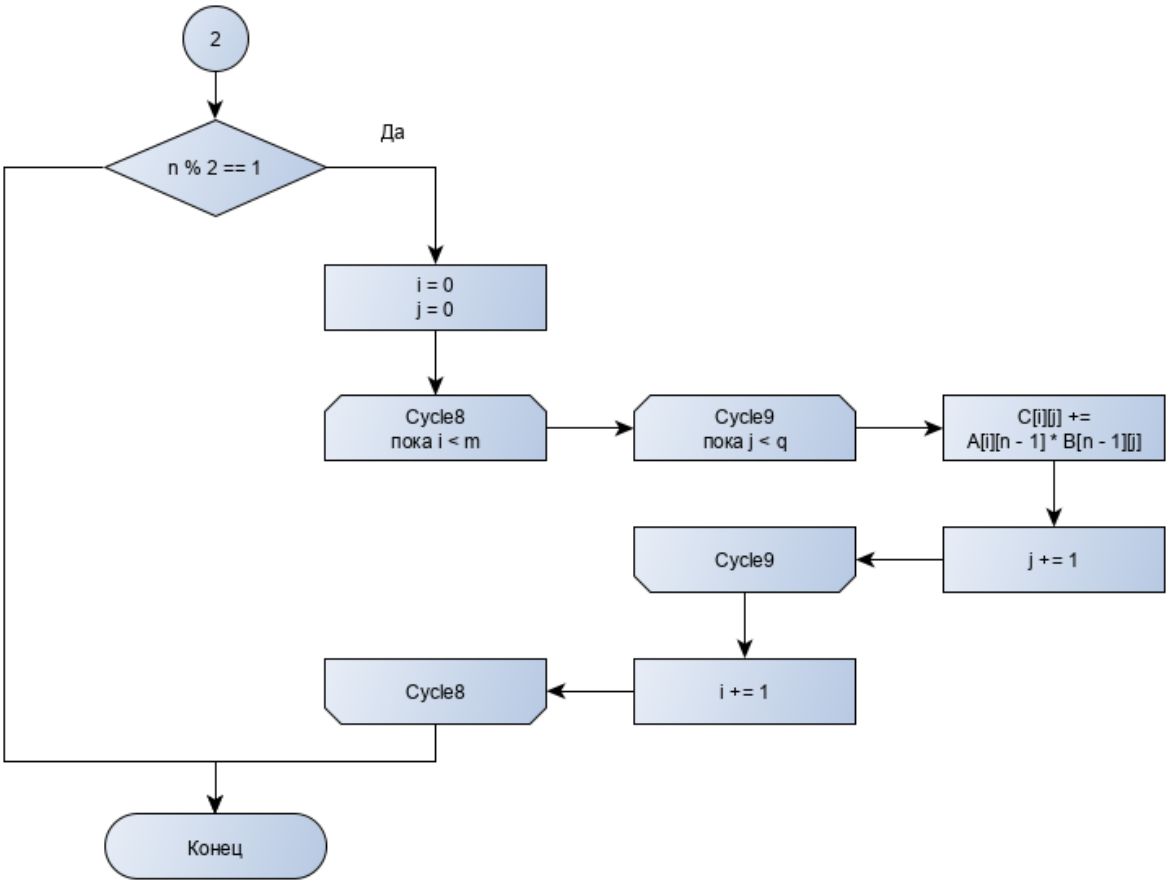
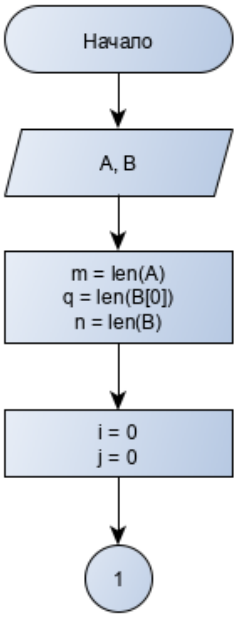
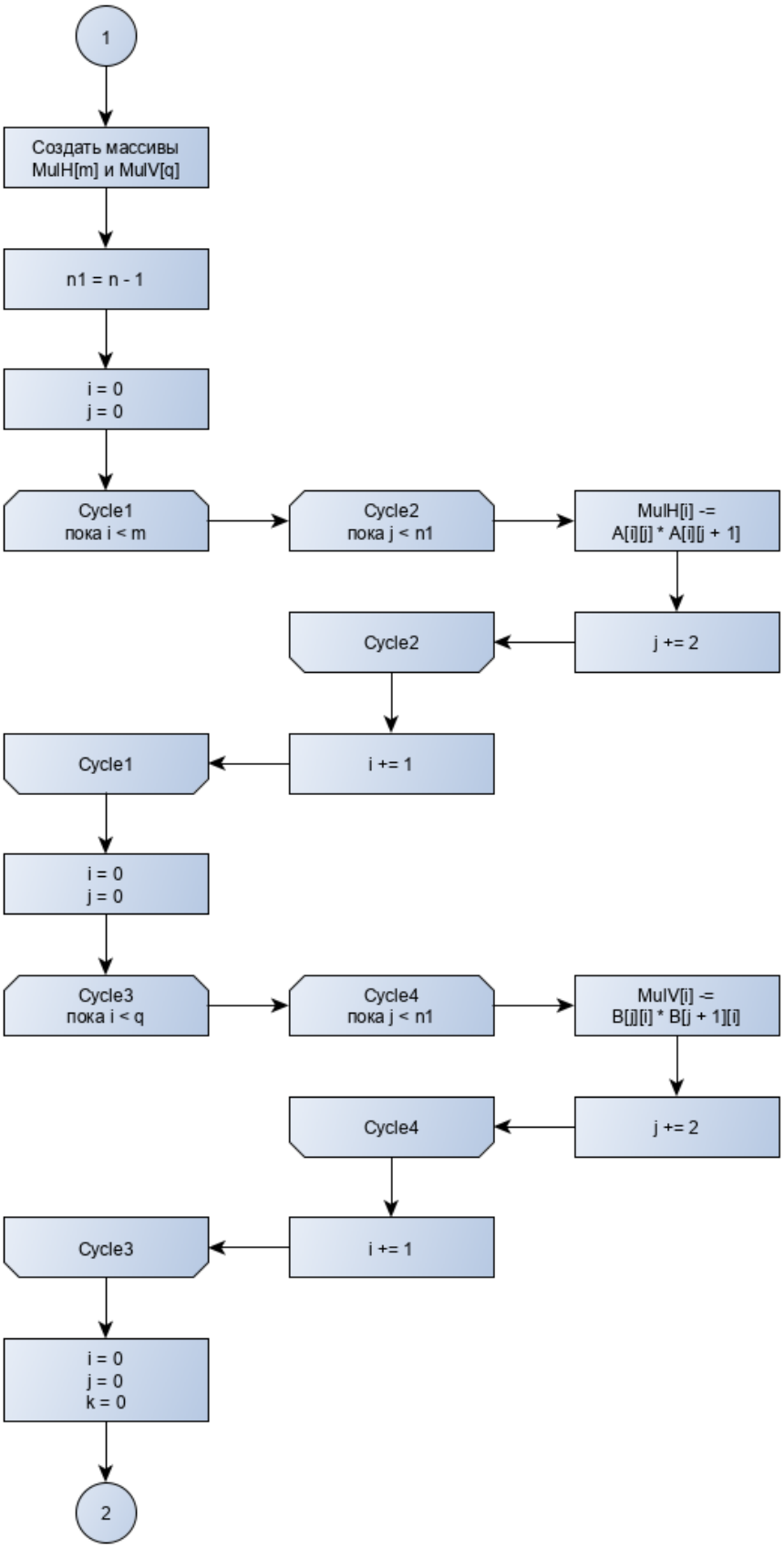


Рисунок 4 – Схема алгоритма Винограда




1
Создать массивы MulH[m] и MulV[q]
n1 = n - 1
i = 0
j = 0
Cycle1 пока i < m
Cycle2 пока j < n1
MulH[i] -= A[i][j] * A[i][j + 1]
j += 2
Cycle2
i += 1
Cycle1
i = 0
j = 0
Cycle3 пока i < q
Cycle4 пока j < n1
MulV[i] -= B[j][i] * B[j + 1][i]
j += 2
Cycle4
i += 1
Cycle3
i = 0
j = 0
k = 0
2

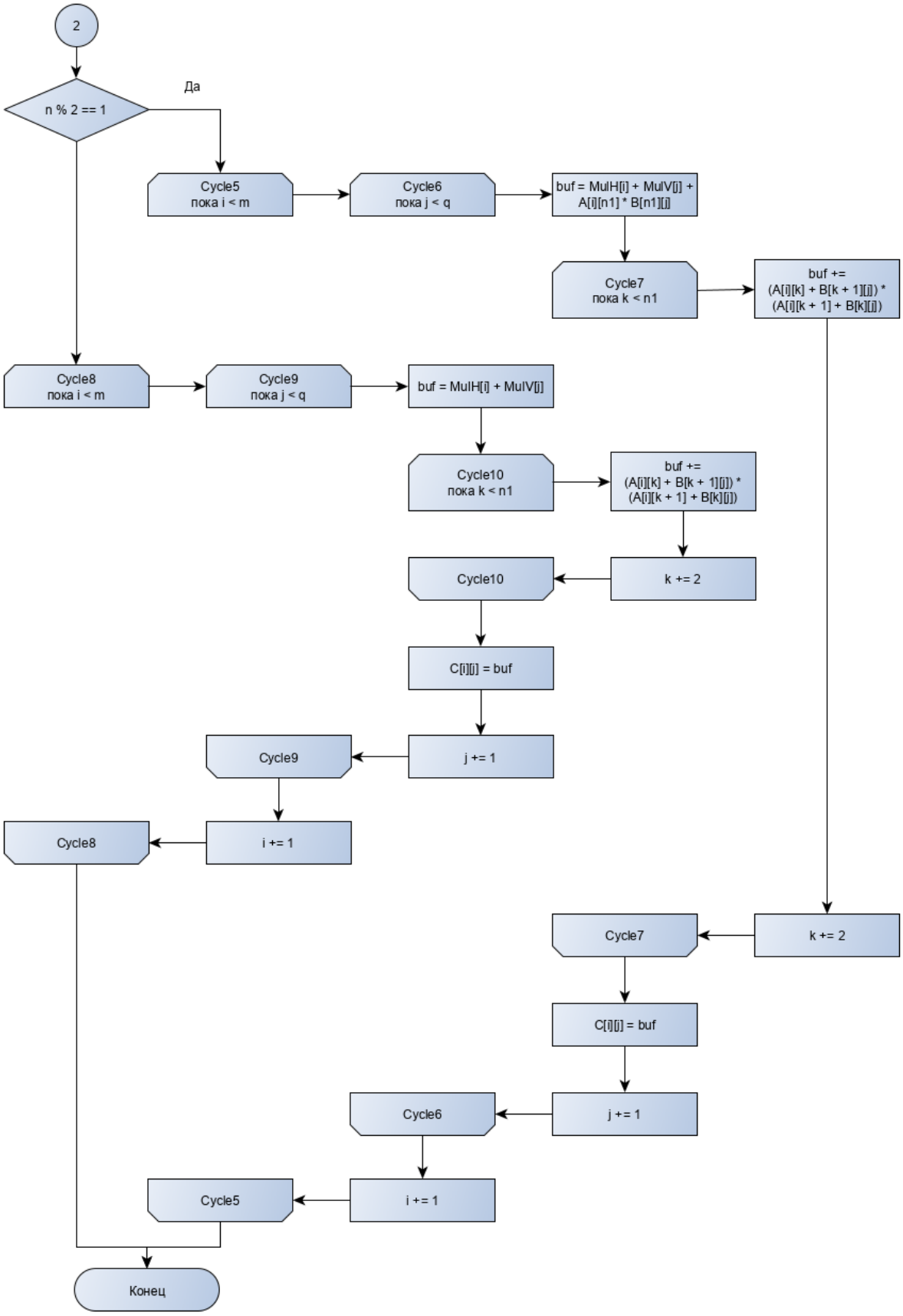


Рисунок 5 – Схема алгоритма Винограда с набором оптимизаций

2.2 Сравнительный анализ алгоритмов

Введем модель вычислений трудоемкости алгоритма. Пусть трудоемкость 1 у следующих операций: +, -, *, /, %, =, ==, !=, <, <=, >, >=, []. Трудоемкость условного перехода примем за 1 (а также трудоемкость самого условия). Трудоемкость цикла: $f_{\text{цикла}} = f_{\text{иниц}} + f_{\text{сравн}} + N * (f_{\text{тела}} + f_{\text{инкрем}} + f_{\text{сравн}})$. Трудоёмкость будет вычисляться по листингу.

Стандартная реализация алгоритма не эффективна по времени, так как обладает трудоемкостью $13MNQ + 4MQ + 4M + 2$, но требует лишь MQ памяти под результат. Алгоритм винограда позволяет улучшить трудоемкость до $\frac{19}{2}MNQ + \frac{11}{2}MN + \frac{11}{2}QN + 11MQ + 9M + 5Q + 8 + \begin{bmatrix} 0 \\ 12MQ + 4M + 2 \end{bmatrix}$, но дополнительно будет использовать $M + Q$ памяти под предварительно вычисляемые значения. А применив к алгоритму ряд оптимизаций можно добиться трудоемкости $8MNQ + 5MN + 5QN + 11MQ + 8M + 4Q + 8 + \begin{bmatrix} 0 \\ 6MQ \end{bmatrix}$ в обмен на использование лишь нескольких дополнительных локальных переменных.

2.3 Вывод

Несмотря на простоту реализации стандартного алгоритма умножения матриц, он делает много лишних операций. Алгоритм Винограда не обладает этим недостатком, при этом требуя совсем немного дополнительной памяти. Но рекомендуется к использованию его модифицированная версия, которая уменьшает в трудоемкости коэффициент при MNQ на 16%.

3. Технологическая часть

В данной части приведены используемые технические средства, а также примеры тестирования и листинг программы.

3.1 Требования к программному обеспечению

Программа должна корректно считывать две матрицы, верно вычислять их произведение. Требуется также обеспечить возможность замера времени работы каждого алгоритма на различных размерах квадратных матриц.

3.2 Средства реализации

Выбран язык программирования Python3 за кроссплатформенность, автоматическое освобождение памяти, высокую скорость разработки. Замер процессорного времени проводился функцией time.process_time() модуля time. Функция представлена на *листинге 1*.

Листинг 1 – Функция замера времени.

```
def get_calc_time(func, matr1, matr2, matr3, it):
    t1 = 0

    for i in range(it):
        t2 = time.process_time()
        func(matr1, matr2, matr3)
        t1 += time.process_time() - t2

    return (t1 / it)
```

3.3 Листинг кода

Исходный код программы приведен на *листингах 2,3,4.*

***Листинг 2* - Стандартный алгоритм умножения матриц**

```
def ord_matr_mul(a, b, c):
    m = len(a)
    q = len(b[0])
    n = len(b)

    for i in range(m):
        for j in range(q):
            for k in range(n):
                c[i][j] = c[i][j] + a[i][k] * b[k][j]

    return c
```

Листинг 3 - **Алгоритм Винограда**

```
def vin_matr_mul(a, b, c):
    m = len(a)
    q = len(b[0])
    n = len(b)
    MulH = [0] * m
    MulV = [0] * q

    for i in range(m):
        for j in range(0, n - 1, 2):
            MulH[i] -= a[i][j] * a[i][j + 1]

    for i in range(q):
        for j in range(0, n - 1, 2):
            MulV[i] -= b[j][i] * b[j + 1][i]

    for i in range(m):
        for j in range(q):
            c[i][j] = MulH[i] + MulV[j]
            for k in range(0, n - 1, 2):
                c[i][j] += (a[i][k] + b[k + 1][j]) * (a[i][k + 1] +
b[k][j])

    if n % 2:
        for i in range(m):
            for j in range(q):
                c[i][j] += a[i][n - 1] * b[n - 1][j]

    return c
```

Листинг 4 - **Алгоритм Винограда с набором оптимизаций**

```
def opt_vin_matr_mul(a, b, c):
    m = len(a)
    q = len(b[0])
    n = len(b)
    MulH = [0] * m
    MulV = [0] * q

    n1 = n - 1

    for i in range(m):
        for j in range(0, n1, 2):
            MulH[i] -= a[i][j] * a[i][j + 1]

    for i in range(q):
        for j in range(0, n1, 2):
            MulV[i] -= b[j][i] * b[j + 1][i]

    if n % 2:
        for i in range(m):
            for j in range(q):
                buf = MulH[i] + MulV[j] + a[i][n1] * b[n1][j]
                for k in range(0, n1, 2):
                    buf += (a[i][k] + b[k + 1][j]) * (a[i][k + 1] +
b[k][j])
                c[i][j] = buf
```

```
    else:
        for i in range(m):
            for j in range(q):
                buf = MulH[i] + MulV[j]
                for k in range(0, n1, 2):
                    buf += (a[i][k] + b[k + 1][j]) * (a[i][k + 1] +
b[k][j])
                c[i][j] = buf

    return c
```

3.4 Описание тестирования

Тестирование проводится по методу чёрного ящика[3]. Требуется проверить корректность работы на квадратных матрицах с четным размером и нечетными размерами сторон, на неквадратных матрицах, а также на умножении вектора на вектор.

3.5 Вывод

Текущая реализация на языке Python позволяет корректно считывать матрицы, вычислять их произведение, а также производить замеры времени для определенного диапазона размеров входных матриц.

4. Экспериментальная часть

В этой части приведены пример интерфейса, входные данные тестирования, результаты замера времени и их анализ.

4.1. Примеры работы

На *рис.* 1, 2 приведены изображения внешнего вида интерфейса программы во время его работы.

```
A:
2 3
1 2 3
4 5 6

B:
3 1
1
2
3

Стандартный алгоритм умножения матриц:
[14]
[32]

Алгоритм Винограда:
[14]
[32]

Алгоритм Винограда с набором оптимизаций:
[14]
[32]
```

Рисунок 1 – Пример работы программы на неквадратных матрицах

```
A:
3 3
2 6 43
34 2 4
2 3 3

B:
3 3
-1 0 32
6 23 1
0 1 -4

Стандартный алгоритм умножения матриц:
[34, 181, -102]
[-22, 50, 1074]
[16, 72, 55]

Алгоритм Винограда:
[34, 181, -102]
[-22, 50, 1074]
[16, 72, 55]

Алгоритм Винограда с набором оптимизаций:
[34, 181, -102]
[-22, 50, 1074]
[16, 72, 55]
```

Рисунок 2 – Пример работы программы на квадратных матрицах

4.2. Результаты тестирования

В *таблице 3* представлены результаты тестирования по методу чёрного ящика [3] в следующем порядке: стандартный алгоритм, алгоритм Винограда, алгоритм Винограда с набором оптимизаций.

Таблица 3

Результаты тестирования по методу черного ящика

Размер матр. A	Размер матр. B	Результат
1 * 6	6 * 8	Ответ верный
4 * 5	5 * 1	Ответ верный
4 * 4	4 * 4	Ответ верный
5 * 5	5 * 5	Ответ верный
3 * 4	4 * 2	Ответ верный

Все тесты пройдены успешно.

4.3. Постановка эксперимента по замеру времени

*З*амер времени проводился для двух квадратных матриц одинаковых размеров. Размер стороны матрицы составлял от 100 до 1000 с шагом 100 при лучшем случае и от 101 до 1001 с шагом 100 при худшем случае. Один эксперимент повторялся не менее 5 раз, результат одного эксперимента рассчитывался как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

4.4. Сравнительный анализ на материале экспериментальных данных

Зависимость времени выполнения реализаций алгоритмов от линейного размера квадратных матриц представлен на *рисунках 4,5*.

Рисунок 4

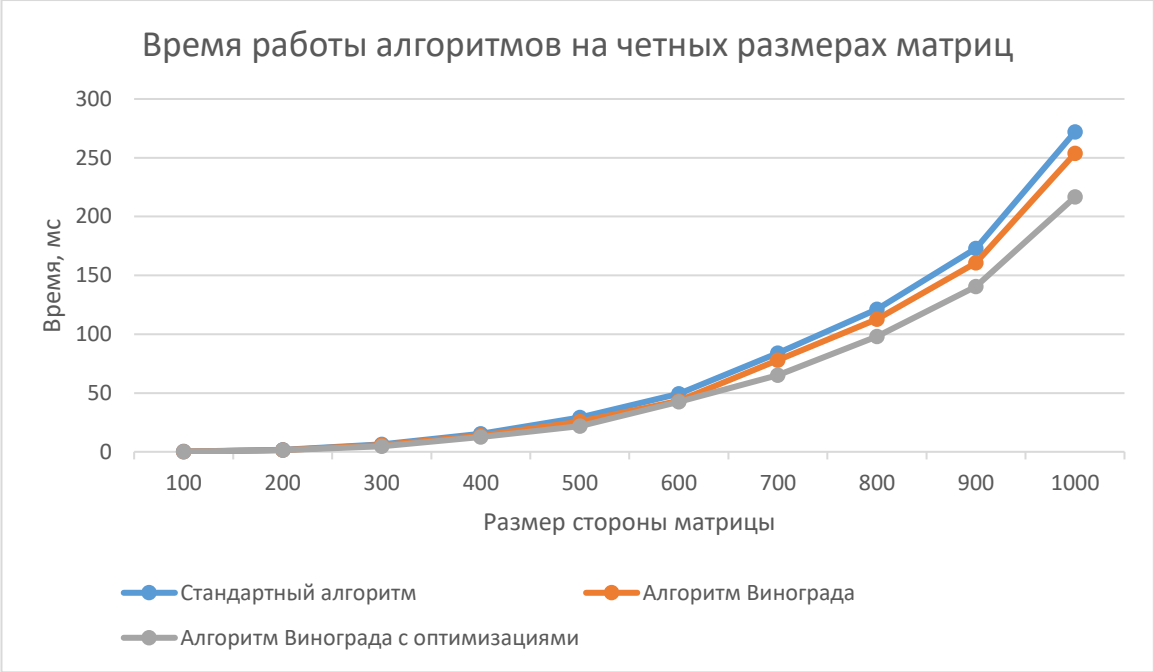
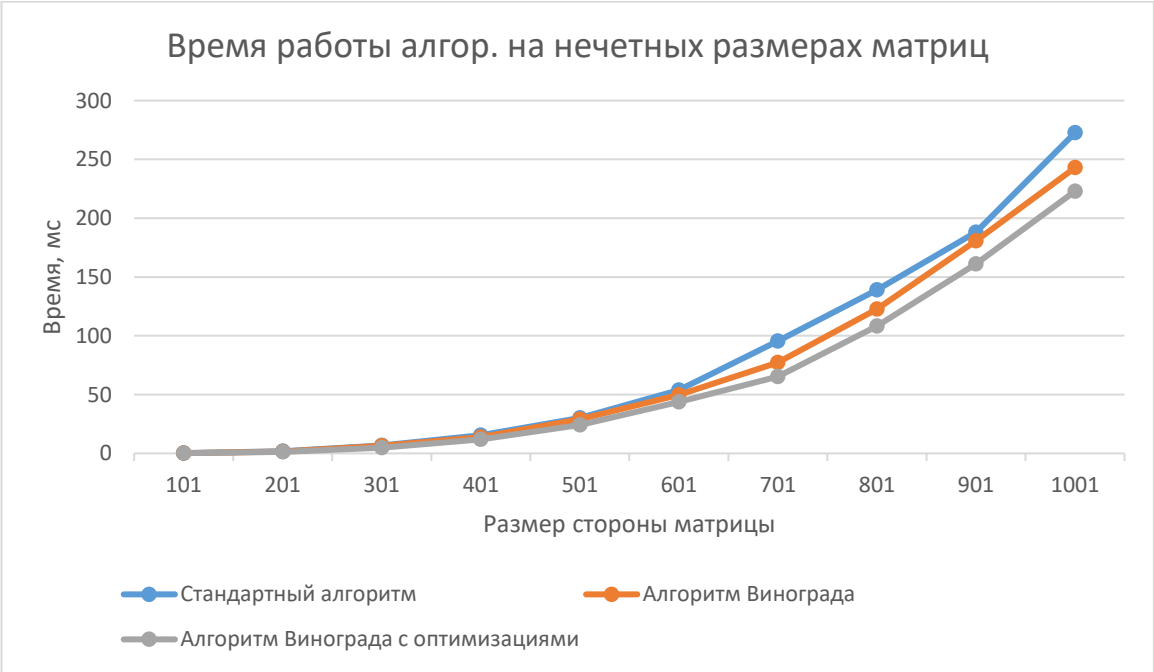


Рисунок 5



4.5 Вывод

По результатам эксперимента подтверждена трудоемкость алгоритмов, указанная в конструкторской части. Стандартный алгоритм работает медленнее на любых размерах квадратных матриц, а оптимизированный алгоритм Винограда – быстрее. Эти результаты объясняются меньшим коэффициентом у слагаемого MNQ в трудоемкости алгоритма [4].

Экспериментально подтверждена сложность алгоритмов. Оптимизированный алгоритм Винограда быстрее на любых размерах матриц. Так на размере 1000 * 1000 он быстрее стандартного алгоритма на 20% и алгоритма Винограда на 15%, что в абсолютных величинах составляет 55.3 сек. и 37 сек. соответственно.

Заключение

В рамках данной работы успешно изучен алгоритм Винограда умножения матриц и пути его оптимизации. Применен метод динамического программирования его реализации. Получены практические навыки реализации указанных алгоритмов: стандартного алгоритма, алгоритма Винограда и алгоритма Винограда с набором оптимизаций. Проведен сравнительный анализ реализаций алгоритмов умножения матриц по затрачиваемым ресурсам (времени и памяти). Подтверждены экспериментально различия во временной эффективности реализаций алгоритмов умножения матриц при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения на варьирующихся размерах матриц. Дано описание и обоснование полученных результатов.

Список литературы

1. time — Time access and conversions // Python URL: https://docs.python.org/3/library/time.html (дата обращения: 1.11.2019).
2. Матричная алгебра в жизни человека // База знаний "Allbest" URL: https://otherreferats.allbest.ru/mathematics/00489400_0.html (дата обращения: 1.11.2019).
3. Kara kutu test technologisi URL:https://www.mobilhanem.com/kara-kutu-test-teknigi-ve-uygulanmasi/ (дата обращения 1.11.2019)
4. Bilgisayar tabanli sistemlerde test otomatizasyonunun tasarlanmasi ve gerçeklenmesi. Hacettepe Üniversitesi: 2015. 70 c. URL:shorturl.wizarmh/2lab
5. Умножение матриц // AlgoLib URL: http://www.algolib.narod.ru/Math/Matrix.html (дата обращения: 9.11.2019).