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»

Отчёт по лабораторной работе №2

По дисциплине: «Анализ алгоритмов»

Тема: «Алгоритмы умножения матриц»

Студент Мередова Айджахан

Группа ИУ7-56Б

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

Москва - 2020 г.

Введение

Матрица - математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам.Две матрицы одинакового размера можно поэлементно сложить или вычесть друг их друга. Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй. Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга. Умножение матрицы A размера $m \times n$ и матрицы B размера $n \times l$ приводит к получению матрицы C размера $m \times l$, каждый элемент которой определяется в соответствии с выражением:

$$C_{ij} = \sum_{p=1}^{m} a_{ip} * b_{pj} \tag{1}$$

1. Аналитическая часть

1.1. Постановка задачи

Цель данной лабораторной работы: Провести сравнительный анализ алгоритмов умножения матриц и получить навык оптимизации алгоритмов. Задачи данной лабораторной работы:

1) Дать математическое описание (формуты рассчётов для алгоритмов);

2) Реализовать старндартный алгоритм умножения матриц и алгоритм Винограда;

3) Разработать оптимизированный алгоритм Винограда;

4) Дать теоретическую оценку трудоёмкости трёх алгоритмов;

5) Провести замеры процессорного времени работы реализаций алгоритмов;

1.2. Описание алгоритма

1.2.1. Последовательный алгоритм умножения матриц

Последовательный алгоритм умножения матриц представляется тремя вложенными циклами.

Листинг 1. Последовательный алгоритм умножения матриц

```
MatrixA[n][n]
MatrixB[n][n]
MatrixC[n][n]
for i in range n:
  for j in range n:
    MatrixC[i][j] = 0
    for k in range n:
      MatrixC[i][j] += MatrixA[i][k] * Matrix[k][j]
```

Этот алгоритм является итеративным и ориентирован на последовательное вычисление строк матрицы C. Поскольку каждый элемент результирующей матрицы есть скалярное произведение строки и столбца исходных матриц, то для вычисления всех элементов матрицы C размером $n \times n$ необходимо выполнить $n^2x(2n-1)$ скалярных операций и затратить время

$$T_1 = n^2 * (2n-1) * \tau, \tag{2}$$

где τ есть время выполнения одной элементарной скалярной операции.

1.2.2. Алгоритм 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. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно:

$$V \times W = v1w1 + v2w2 + v3w3 + v4w4. \tag{3}$$

Это равенство можно переписать в виде:

$$V \times W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4. \tag{4}$$

Кажется, что выражение (4) задает больше работы, чем (3): 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.

1.2.3. Вывод

Были рассмотрены алгоритмы последовательного умножения матриц и алгоритм Винограда. Их разница заключается в предварительной обработке скалярного произведения соответствующих строк и столбцов исходных матриц, а также уменьшение количества операций умножения.

1.3. Вычисление сложности алгоритма

Расммотрим сложности операций. **Операции со сложностью "1":**

1) Присваивание «=»

2) Сложение «+»

3) Вычитание «-»

4) Унарный плюс «+»

5) Унарный минус «-»

6) Умножение «*»

7) Деление «/»

8) Взятие остатка от деления «%»

9) Инкремент (префиксный и постфиксный) «++»

10) Декремент (префиксный и постфиксный) «- -»

11) Индексация (обращение к элементу массива) «[]»

12) Присваивание со сложением «+=»

13) Равенство «==»

14) Неравенство «!=»

15) Больше « > »

16) Меньше « < »

17) Больше или равно « >= »

18) Меньше или равно « <= »

19) Логическое отрицание « ! »

20) Логическое умножение « && »

21) Логическое сложение «||»

22) Побитовая инверсия « ~ »

23) Побитовое И «&»

24) Побитовое ИЛИ « | »

25) Присваивание со сложением « += »

26) Присваивание с вычитанием « -= »

27) Присваивание с умножением « *= »

28) Присваивание с делением « /= »

29) Присваивание со взятием остатка от деления « % = »

Условный оператор:
if(условие)
{
// тело условия A
}
else
{
// тело условия B
}

Пусть стоимость перехода к одной из ветвей решения равна 0, тогда трудоёмкость if при $f_(min) = min(f_a, f_b), f_(max) = max(f_a, f_b)$, получим

$$f_(if) = f + \begin{cases} D(S_1[1..i], S_2[1..j-1]) + 1 \\ D(S_1[1..i-1], S_2[1..j]) + 1 \end{cases} \quad (5)$$

Цикл со счётчиком:
for (int i =0 ; i < n ; i++)
{
//тело цикла
}

Начальная инициализация int i = 0 выполняется всего один раз - 1 операция. Условие $i < n$ проверяется перед каждой итерацией и при входе в цикл - n + 1 операций. Тело цикла выполняется n раз - $n * f$. Изменение счётчика $i++$ выполняется на каждой итерации, но перед проверкой условия - n операций. Итого сложность цикла со счётчиком:$2 + n * (2 + f)$, где f - сложность тела цикла.

2. Конструкторская часть.

2.1. Схемы алгоритмов.

На рисунках 1 - 4 представлены схемы алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

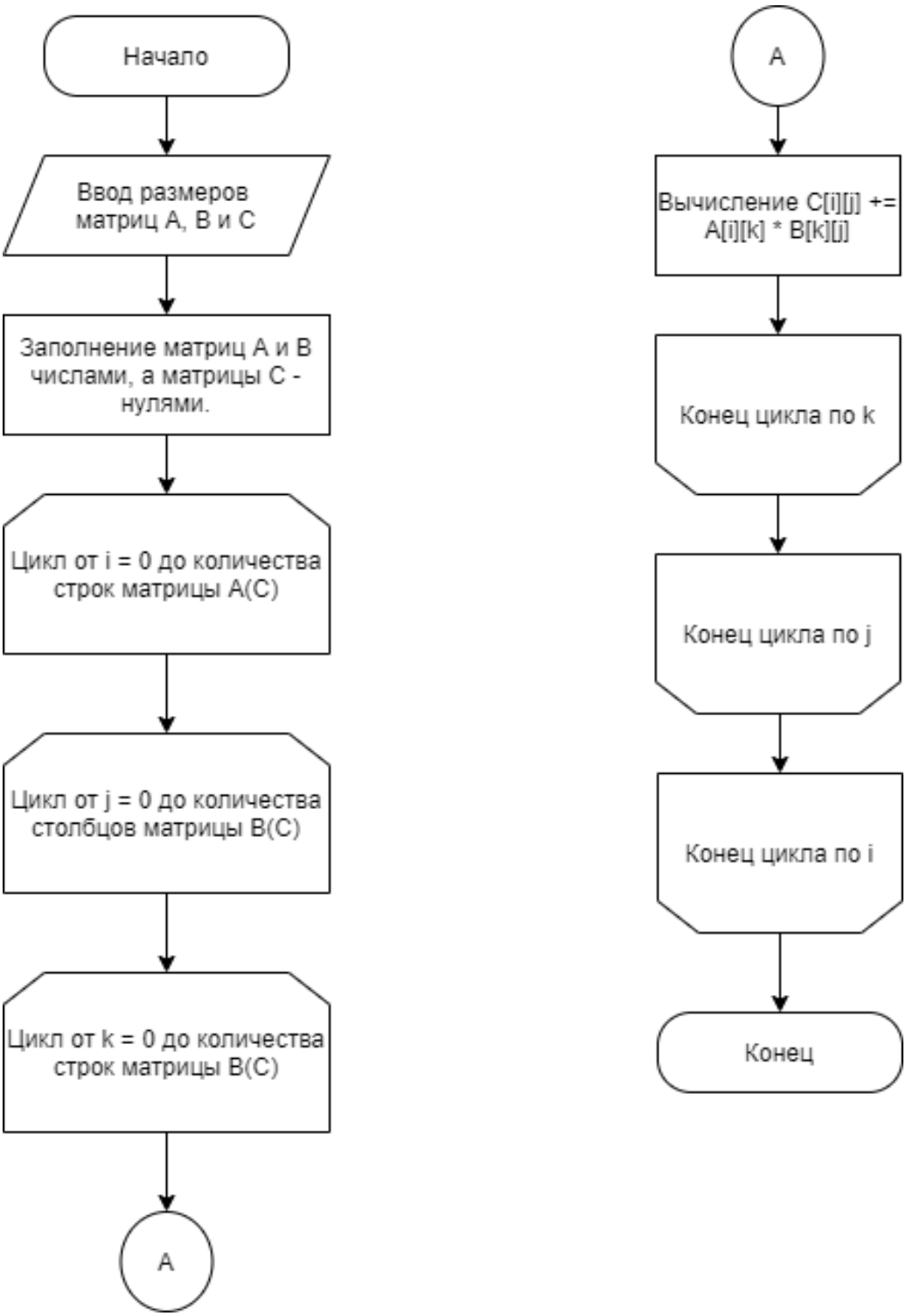

Рис. 1. Схема стандартного алгоритма умножения матриц

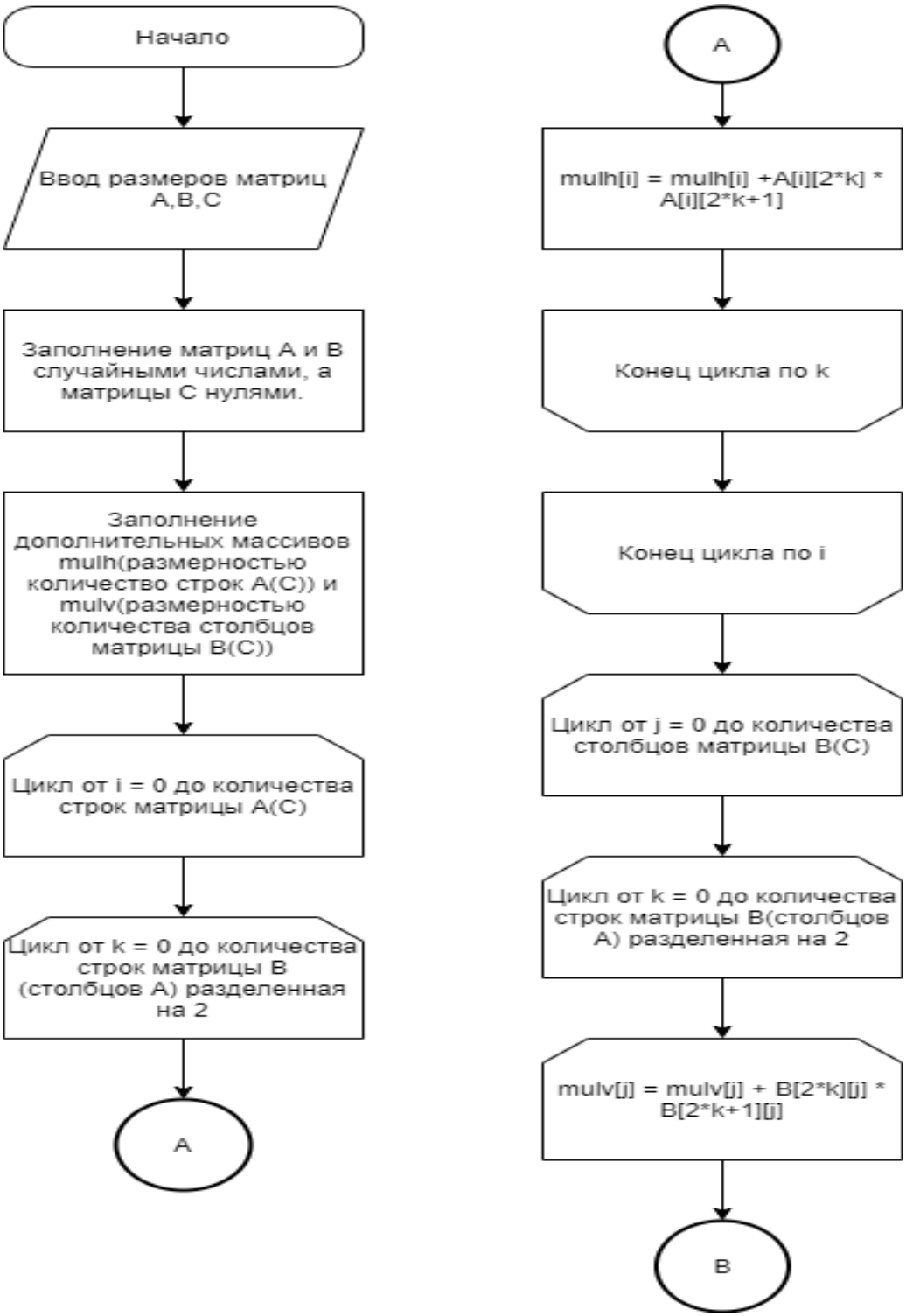

Рис. 2. Схема алгоритма Винограда умножения матриц(1)

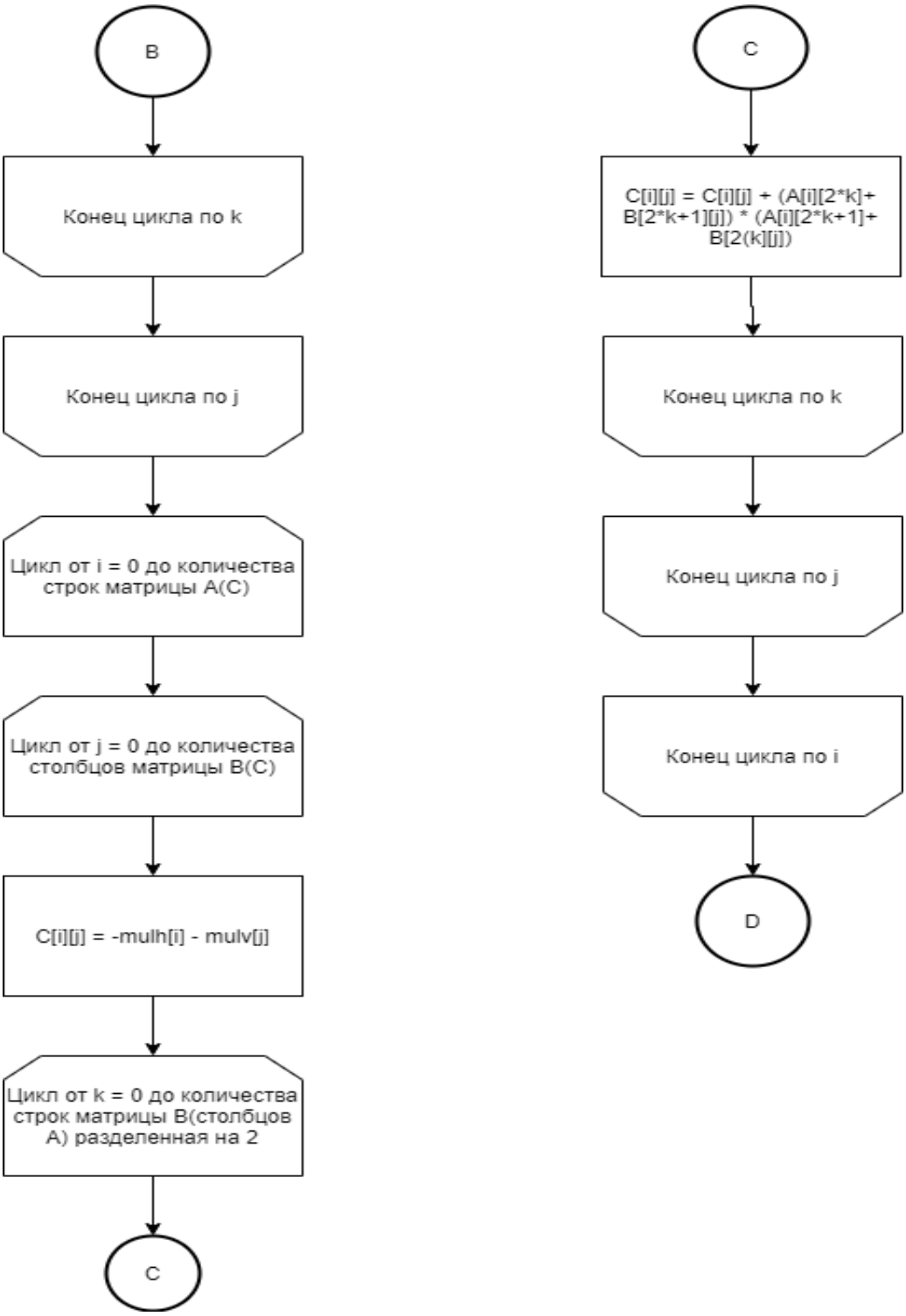

Рис. 3. Схема алгоритма Винограда умножения матриц(2).

Рис. 4. Схема алгоритма Винограда умножения матриц(3)

3. Технологическая часть

3.1. Выбор языка программирования

Python[1] — это высокоуровневый язык программирования, который используется в различных сферах IT, таких как машинное обучение, разработка приложений, web, парсинг и другие. С ним легко работать, что сокращает время разработки. Написанный в удобочитаемом формате, Python делает процесс разработки программного обеспечения быстрым, удобным и максимально упрощенным. По сравнению с другими языками, Python в 5-10 раз быстрее по времени разработки, однако медленный при выполнении программ. Он обеспечивает расширенные возможности управления процессами и объектно-ориентированный дизайн, помогая как в скорости, так и в производительности.

3.2. Методы замера времени в программе

3.2.1. Время

Сушествует несколько способов измерения процессорного времени исполнения программы. Помимо стандратного модуля *time* есть библиотека *timeit* [2]. Этот модуль предоставляет простой способ найти время выполнения маленьких битов кода Python. *timeit* запускает фрагмент кода миллионы раз(значение по умолчанию - 1000000), так что получаем наиболее статистически значимое измерение времени выполнения кода.

3.2.2. Улучшение точности замеров времени

Чтобы получить более точные результаты, каждый тест запускается несколько раз, все полученные значения времени(в тиках) суммируются и делятся на количество запусков кода. Таким образом, получаем среднее время выполнения кода.

4. Экспериментальная часть

4.1. Листинг кода

В листингах 3 - представлена реализация стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

Листинг 2. Стандартный алгоритм умножения матриц

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

Листинг 3. Алгоритм Винограда умножения матриц

```
  def vinograd(a, b, c, n, m, q):
    mulh = [0 for i in range(n)]
    mulv = [0 for i in range(q)]

    for j in range(q):
      for k in range(m // 2):
        mulv[j] = mulv[j] + (b[2*k][j] * b[2*k+1][j])


    for i in range(n):
      for k in range(m // 2):
        mulh[i] = mulh[i] + (a[i][2*k] * a[i][2*k+1])

    for i in range(m):
      for j in range(q):
        c[i][j] = - mulh[i] - mulv[j]
        for k in range(m // 2):
          c[i][j] = c[i][j] + (a[i][2*k] + b[2*k+1][j]) * (a[i][2*
              k+1] + b[2*k][j])

    if m % 2 != 0:
      for i in range(n):
        for j in range(q):
          c[i][j] = c[i][j] + a[i][m-1] * b[m-1][j]
```

Листинг 4. Оптимизированный алгоритм Винограда умножения матриц

```
def isOdd(a, b, m, i, j):
  if m % 2 == 0:
    return 0
  return a[i][m-1] * b[m-1][j]

def optimize_vinograd(a, b, c, n, m, q):
```

```
    mulh = [0 for i in range(n)]
    mulv = [0 for i in range(q)]

    d = m // 2
    for j in range(q):
      for k in range(d):
        mulv[j] += (b[2*k][j] * b[2*k+1][j])


    for i in range(n):
      for k in range(d):
        mulh[i] += (a[i][2*k] * a[i][2*k+1])


    temp = 0
    for i in range(n):
      for j in range(q):
        temp = isOdd(a, b, m, i, j)
        temp -= mulh[i] + mulv[j]
        for k in range(0,d, 2):
          temp += (a[i][2*k] + b[2*k+1][j]) * (a[i][2*k+1] + b[2*k][
              j])
        c[i][j] = temp
```

4.2. Примеры работы

На таблице 1 приведены примеры работы алгоритмов умножения матриц.

A: |-20 -7| |-19 -11| B: |-2 -7 | | 6 -6 |	C: |-2 182| |-28 199|	C: |-2 182 | |-28 199 |	C: |-2 182| |-28 199|
A: | -6 -6| |18 14| |5 4| B: |-16| |-14|	C: |180 | |-484| |-136|	C: | 180 | |-484 | |-136 |	C: | 180 | |-484 | |-136 |
A: |-8 3 18| |-13 18 0 | B: |13 3| |-15 17| |1 14|	C: |-131 279| |-439 267|	C: |-131 279| |-439 267|	C: |-131 279| |-439 267|
A: | 9 | |-16 | | -3 | | 11 | B: | 1 16 15 |	C: | 9 144 135 | |-16 -256 -240 | |-3 -48 -45 | |11 176 165 |	C: |9 144 135| |-16 -256 -240| |-3 -48 -45 | |11 176 165|	C: |9 144 135| |-16 -256 -240| |-3 -48 -45 | |11 176 165|

4.3. Оценка трудоёмкости

1) Стандартный алгоритм: $f_{std} = 2+N(2+2+Q(2+2+M(2+1+8+1+1))) = 13NMQ + 4NQ + 4N + 2 \approx 13NMQ$

2) Алгоритм Винограда:

$$f_{v1+} = 2+N(2+2+3+\frac{N}{2}(3+1+6+2+3+)) = \frac{15}{2}MN+7N+2 \approx \frac{15}{2}MN \quad (6)$$

$$f_{v1} = 2+N(2+2+2+\frac{N}{2}(2+5+1+1+1)) = 10NM+6N+2 \approx \frac{10}{2}NM = 5NM \quad (7)$$

$$f_{v2} = 2+Q(2+2+3+\frac{N}{2}(3+6+1+2+3)) = \frac{15}{2}QM+7Q+2 \approx \frac{15}{2}QM \quad (8)$$

$$f_{v2+} = 2+Q(2+2+\frac{Q}{2}(2+5+1+1+1)) = 10QM+6Q+2 \approx \frac{10}{2}QM = 5QM \quad (9)$$

$$f_{v3} = 2+N(2+2+Q(2+7+3+\frac{N}{2}(3+12+1+5+5))) = \frac{26}{2}NMQ+12NQ+4N+2 \approx \frac{26}{2}$$ (10)

Лучший случай 0,
Худший случай f_{v4}

$$f_{v4} = 3 + N(2 + 2 + Q(2 + 13)) = 15QN + 4N + 3 \approx 15QN \quad (11)$$

В формуле (12) внутри цикла внесен f_{v4} из формулы (11). Поэтому появляются лучший и худший случаи.

$$fv3+ = 3+2+N(2+2+Q(2+6+\begin{bmatrix} \text{л.с.}, 0 \\ \text{х.с.}, 6+2+1+1 \end{bmatrix}+2+\frac{N}{2}(2+10+1+2+2+1))) \quad (12)$$

В худшем случае = 9NMQ + 10NQ + 4N + 5
В лучшем случае ≈ 8NMQ

$$f_v = f_{v1} + f_{v2} + f_{v3} + f_{v4} = \frac{15}{2}NM + \frac{15}{2}QM + \frac{26}{2}NMQ + 15QN \quad (13)$$

$$f_{v+} = f_{v1+} + f_{v2+} + f_{v3+} + f_{v4+} = 5NM + 5QM + \begin{bmatrix} \text{л.с.}, 9NMQ + 10NQ \\ \text{х.с.}, 9NMQ + 20NQ \end{bmatrix} \quad (14)$$

Список оптимизации:

1) замена операций = на + = или − = ;

2) избавление от деления в условиях цикла ;

3) занесение проверки на нечетность количества строк внутрь основных циклов;

4) расчет условия для последнего цикла один раз, а далее использование флага;

4.4. Сравнение времени работы

На рисунках 5 - 6 представлены графики сравнения времени работы алгоритма на матрицах разных размеров.

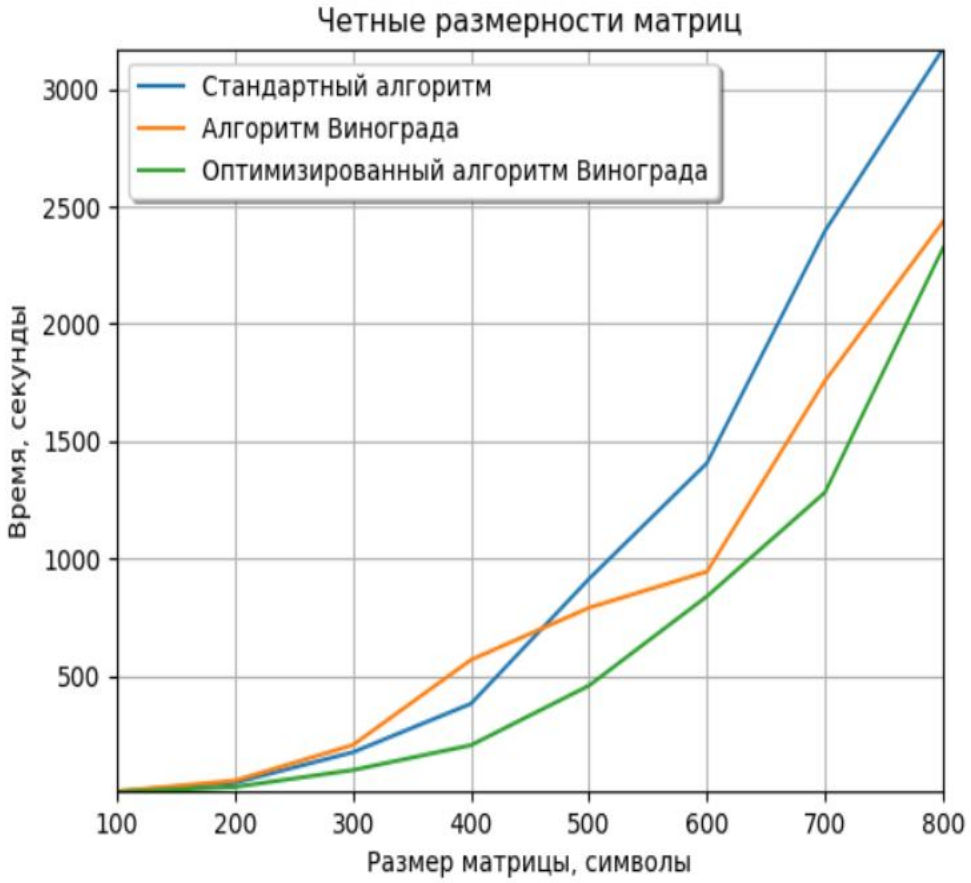

Рис. 5. Сравнение реализаций алгоритмов нахождения произведения матриц при четных размерностях.

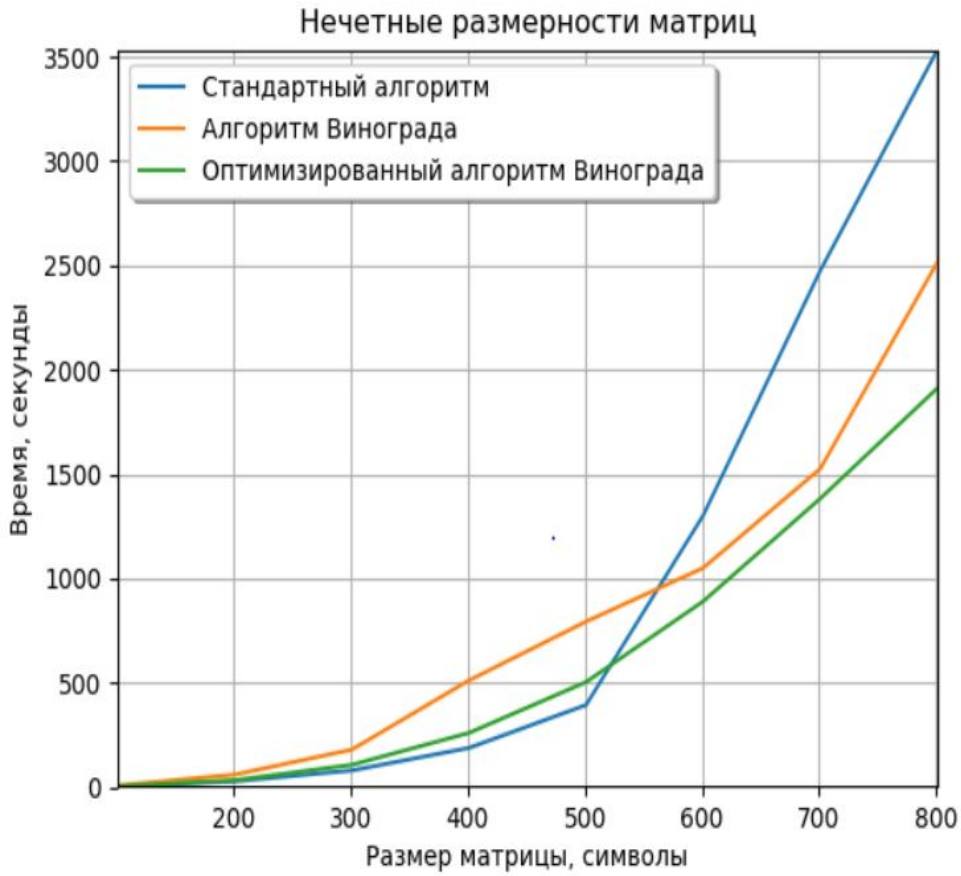

Рис. 6. Сравнение реализаций алгоритмов нахождения произведения матриц при нечетных размерностях.

4.5. Вывод

В результате проведенных экспериментов были получены следующие выводы:

1) оптимизированный алгритм Винограда работает быстрее классического алгоритма;

2) оптимизированный алгорити Винограда работает быстрее обычного алгоритма Винограда;

3) по результатам тестирования на матрицах с нечётными размерностями видно, что неоптимизированный алгоритм Винограда работает медленнее, чем при работе с чётными размерностями;

Заключение

В данной лабораторной работе были реализованы и проанализированы три алгоритма умножения матриц:

- стандартный алгоритм умножения матриц;
- алгоритм Винограда;
- оптимизированный алгоритм Винограда;

В данной лабораторной работе была достигнута цель и были выполнены следующие задачи:

1) были описаны формулы расчета для двух алгоритмов: стандартного и алгоритма Винограда;
2) были реализованы стандартный алгоритм умножения матриц и алгоритм Винограда;
3) был реализован оптимизированный алгоритм Винограда;
4) посчитана теоретическая оценка сложности трех алгоритмов;
5) проведены замеры процессорного времени работы реализаций трех алгоритмов при четных и нечетных размерностях;

Список литературы

[1] Python:что нужно знать. [ЭЛ. РЕСУРС]
Режим доступа: https://skillbox.ru/media/code/story_buzunov/
(Дата обращения: 30.10.2020)

[2] Timeit в Python с примерами. [ЭЛ. РЕСУРС]
Режим доступа: http://espressocode.top/timeit-python-examples/
(Дата обращения: 30.10.2020)

[3] Умножение матриц. [ЭЛ. РЕСУРС]
Режим доступа: http://algolib.narod.ru/Math/Matrix.html
(Дата обращения: 30.10.20)

[4] Знай сложности алгоритмов. [ЭЛ. РЕСУРС]
Режим доступа: https://habr.com/ru/post/188010/
(Дата обращения: 30.10.20)

[5] Параллельные методы матричного умножения. [ЭЛ.РЕСУРС]
Режим доступа: https://intuit.ru/studies/courses/2065/190/lecture/4954
(Дата обращения: 30.10.20)