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

Группа ИУ7-51Б

Тема работы **Алгоритмы умножения матриц**

Студент Баранов Николай Алексеевич

Преподаватель Волкова Лилия Леонидовна

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе рассматриваются алгоритмы перемножения двух матриц: стандартный алгоритм и алгоритм Винограда.

Цель работы: описание и исследование стандартного алгоритма перемножения матриц, а также алгоримта Винограда.

1) описать стандартный алгоритм и алгоритм Винограда, оптимизации к ним, а также модель вычислений, по которой будет сравниваться их трудоемкость;
2) написать программу, реализующую эти алгоритмы в двух версиях (с оптимизациями и без), и рассчитать трудоемкость выполнения по модели вычислений;
3) выбрать инструменты для замера процессорного времени выполнения реализаций алгоритмов;
4) провести анализ затрат реализаций алгоритмов по времени и сравнить с расчетами, полученными по модели вычислений.

1 Аналитическая часть

1.1 Модель вычислений

Модель вычислений описывает правила для задания оценки ресурсной эффективности алгоритмов.

Трудоёмкость следующих операций принимается за 1: $=$, $+$, $-$, $+=$, $-=$, $==$, $!=$, $<$, $<=$, $>$, $>=$, [], $<<$, $>>$, $<<=$, $>>=$, &&, ||, &, |, ^, $\&=$, $|=$.

Трудоёмкость следующих операций принимается за 2: $*$, $/$, $\%$, $*=$, $/=$, $\%=$.

Трудоёмкость цикла рассчитывается следующим образом: пусть имеется цикл for (init; cond; inc) { body }, где init – блок инициализации переменных, cond – блок проверки условия, inc – блок изменения переменных, body – тело цикла. Пусть цикл выполнился N раз. Тогда трудоёмкость цикла f_{for} рассчитывается по формуле 1.1

$$f_{for} = f_{init} + f_{cond} + N * (f_{cond} + f_{inc} + f_{body}) \tag{1.1}$$

где f_{init} – трудоёмкость блока инициализации, f_{cond} – трудоёмкость блока проверки условия, f_{inc} – трудоёмкость блока изменения переменных, f_{body} – трудоёмкость тела цикла.

Трудоёмкость условного оператора рассчитывается следующим образом: пусть имеется условный оператор if (cond) then { body1 } else { body2 }, где cond – блок проверки условия, body1 – тело при выполнении условия, body2 – тело при невыполнении условия. Тогда трудоёмкость условного оператора f_{if} рассчитывается по формуле 1.2

$$f_{if} = f_{cond} + \begin{cases} f_{body1} & \text{; условие cond выполнилось} \\ f_{body2} & \text{; иначе} \end{cases}, \tag{1.2}$$

где f_{cond} – трудоёмкость блока проверки условия, f_{body1} – трудоёмкость тела при выполнении условия, f_{body2} – трудоёмкость тела при невыполнении условия.

Операцией выделения памяти пренебрегаем.

1.2 Алгоритмы перемножения матриц

1.2.1 Стандартный алгоритм

Пусть даны матрица A размером N на M (формула 1.3) и матрица B размером M на K (формула 1.4).

$$A_{NM} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{pmatrix} \tag{1.3}$$

$$B_{MK} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1K} \\ b_{21} & b_{22} & \dots & b_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M1} & b_{M2} & \dots & b_{MK} \end{pmatrix} \tag{1.4}$$

Тогда матрица C размером N на K (формула 1.5), где c_{ij} – элемент матрицы в строке i и столбце j – рассчитывается по формуле 1.6, называется произведением матриц A и B.

$$C_{NK} = \begin{pmatrix} 11 & c_{12} & \dots & c_{1K} \\ c_{21} & c_{22} & \dots & c_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & \dots & c_{NK} \end{pmatrix} \tag{1.5}$$

$$\mathrm{c}_{ij} = \sum_{k=1}^{M} a_{ik} * b_{kj} \tag{1.6}$$

Из определения следует, что количество столбцов в первой матрице должно совпадать с количеством строк во второй матрице, в противном случае произведение невозможно получить. Стандартный алгоритм реализует эту формулу.

1.2.2 Алгоритм Винограда

В случае четного M c_{ij} можно вычислить по формуле 1.7 [1].

$$c_{ij} = \sum_{k=1}^{\frac{M}{2}}((a_{i(2k-1)}+b_{(2k)j})*(a_{i(2k)}+b_{(2k-1)j}))-\sum_{k=1}^{\frac{M}{2}} a_{i(2k-1)}*a_{i(2k)}-\sum_{k=1}^{\frac{M}{2}} b_{(2k-1)j}*b_{(2k)j} \tag{1.7}$$

Для нечетного M достаточно добавить недостающее произведение.

Идея алгоритма Винограда заключается в сокращении количества операций умножения для увеличения скорости вычислений. Суммы во втором и третьем слагаемом можно посчитать заранее и переиспользовать для всех элементов строки или столбца. В таком случае для каждого c_{ij} используется почти в 2 раза меньше операций умножения.

1.3 Оптимизации

Для каждого алгоритма будут рассмотрены следующие оптимизации:

1) замена умножения на 2 на двоичный сдвиг;
2) использование оператора сложения с присваиванием (+=);
3) вынос начальной итерации из каждого внешнего цикла.

1.4 Вывод

В данном разделе была описана модель вычислений, рассмотрены стандартный алгоритм и алгоритм Винограда умножения матриц, а также оптимизации к ним.

2 Конструкторская часть

2.1 Схемы алгоритмов

На вход каждому из алгоритмов подаются матрицы A и B, а также числа N, M и K. Предполагается, что входные данные корректны: N, M и K больше 0, A размером N на M, B размером M на K.

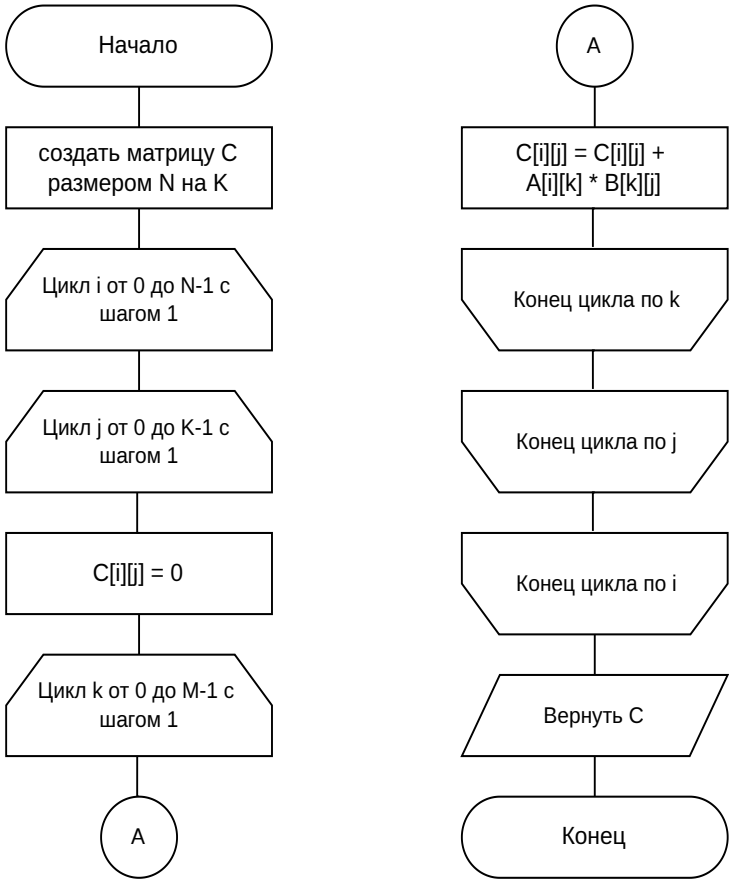

Рисунок 2.1 — Схема стандартного алгоритма нахождения прозведения матриц

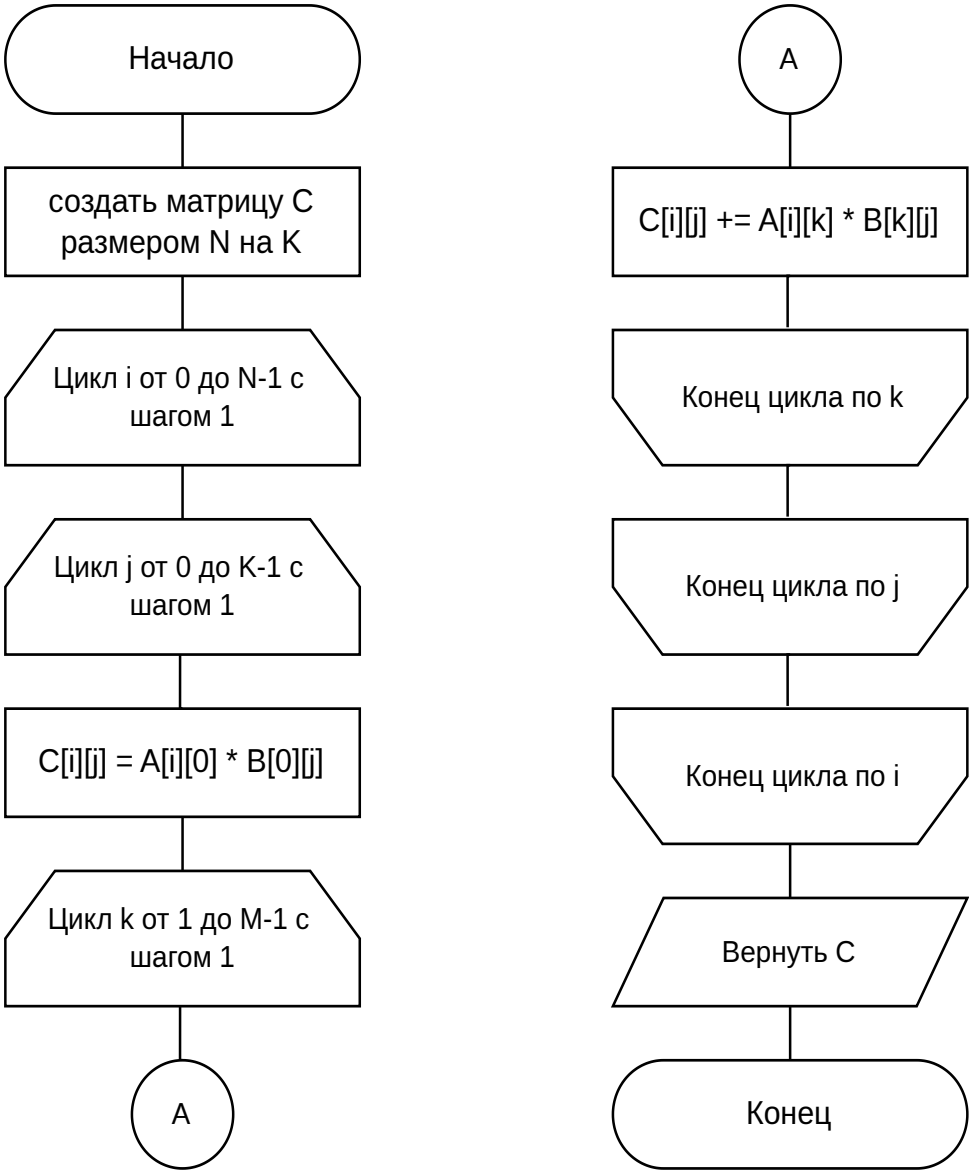

Рисунок 2.2 — Схема стандартного алгоритма нахождения прозведения матриц с оптимизациями

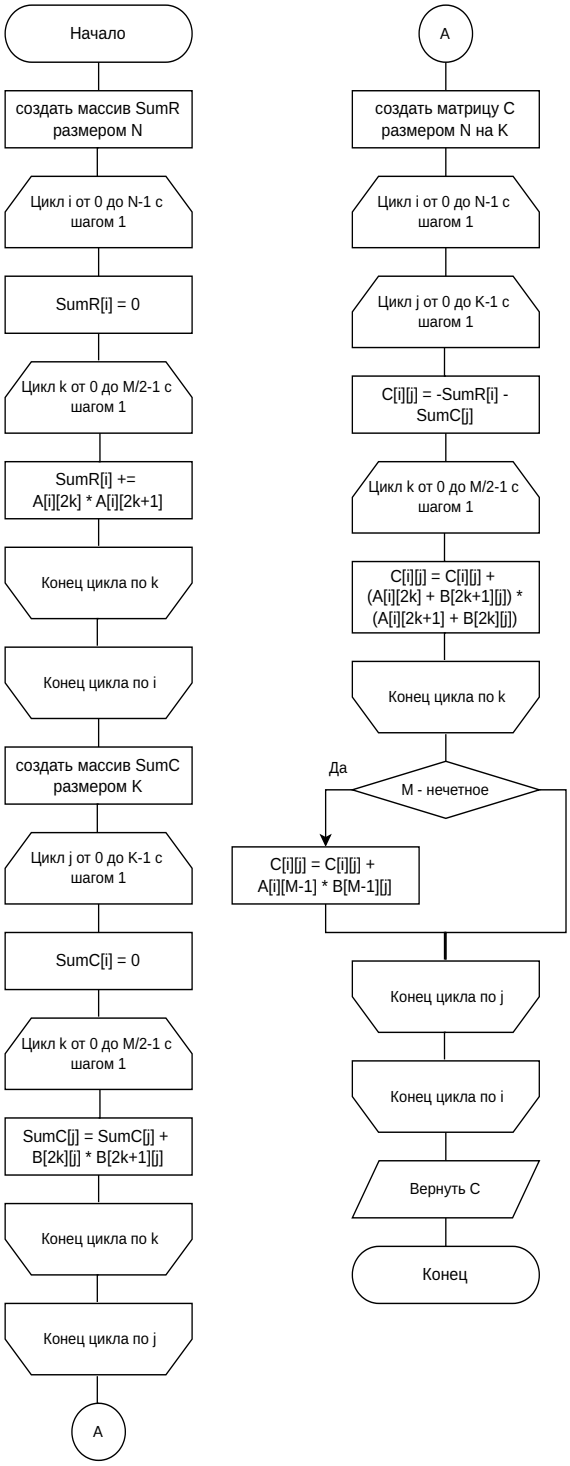

Рисунок 2.3 — Схема алгоритма Винограда

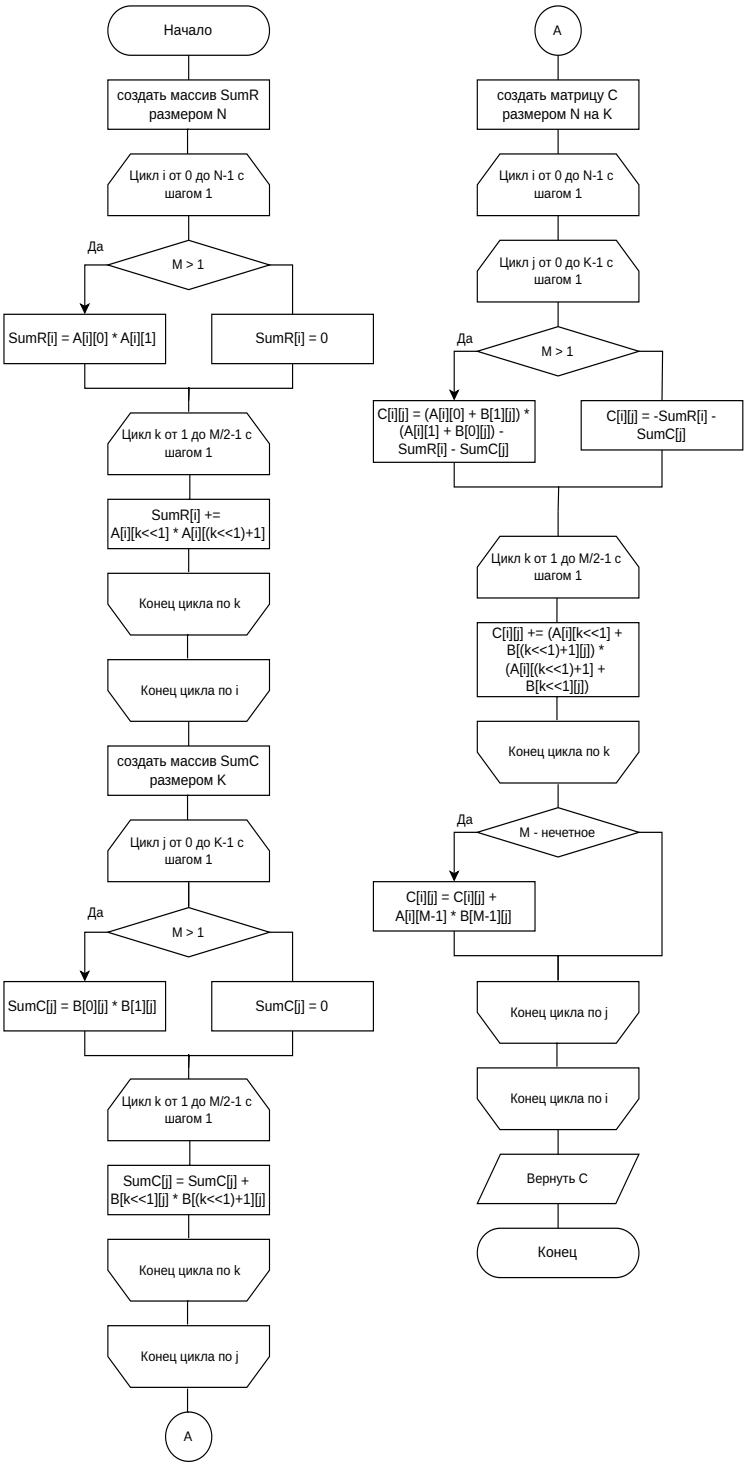

Рисунок 2.4 — Схема алгоритма Винограда с оптимизациями

2.2 Используемые типы и структуры данных

При реализации алгоритмов использованы следующие структуры данных:

— матрица – двумерный массив целых чисел.

2.3 Выводы

В данном разделе были построены схемы алгоритмов и выбраны структуры данных.

3 Технологическая часть

3.1 Средства реализации

Для реализации данной лабораторной работы был выбран язык C++, так как он содержит необходимые средства для реализации алгоритмов.

3.2 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3, 3.4 представлены реализации алгоритмов перемножения матриц.

Листинг 3.1 — Стандартный алгоритм перемножения матриц

```
Matrix StandartAlgorithm::execute() {
  auto c = initMatrix(m, q);
  for (auto i = 0; i < m; ++i) {
    for (auto j = 0; j < q; ++j) {
      c[i][j] = 0;
      for (auto k = 0; k < n; ++k) {
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
      }
    }
  }
  return c;
}
```

Листинг 3.2 — Стандартный алгоритм с оптимизациями

```
Matrix StandartOptimisedAlgorithm::execute() {
  auto c = initMatrix(m, q);
  for (auto i = 0; i < m; ++i) {
    for (auto j = 0; j < q; ++j) {
      c[i][j] = a[i][0] * b[0][j];
      for (auto k = 1; k < n; ++k) {
        c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

```
  return c;
}
```

Листинг 3.3 — Алгоритм Винограда

```
Matrix VinogradAlgorithm::execute() {
  auto c = initMatrix(m, q);
  for (auto i = 0; i < m; ++i) {
    mulR[i] = 0;
    for (auto k = 0; k < n / 2; ++k) {
      mulR[i] = mulR[i] + a[i][2 * k] * a[i][2 * k + 1];
    }
  }
  for (auto j = 0; j < q; ++j) {
    mulC[j] = 0;
    for (auto k = 0; k < n / 2; ++k) {
      mulC[j] = mulC[j] + b[2 * k][j] * b[2 * k + 1][j];
    }
  }
  for (auto i = 0; i < m; ++i) {
    for (auto j = 0; j < q; ++j) {
      c[i][j] = -mulR[i] - mulC[j];
      for (auto k = 0; k < n / 2; ++k) {
        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 (n % 2 == 1) {
        c[i][j] = c[i][j] + a[i][n - 1] * b[n - 1][j];
      }
    }
  }
  return c;
}
```

Листинг 3.4 — Алгоритм Винограда с оптимизациями

```
Matrix VinogradOptimisedAlgorithm::execute() {
  auto c = initMatrix(m, q);
  for (auto i = 0; i < m; ++i) {
    if (n > 1) {
      mulR[i] = a[i][0] * a[i][1];
    } else {
```

```
      mulR[i] = 0;
    }
    for (auto k = 1; k < n / 2; ++k) {
      mulR[i] += a[i][k >> 1] * a[i][(k >> 1) + 1];
    }
  }
  for (auto j = 0; j < q; ++j) {
    if (n > 1) {
      mulC[j] = b[0][j] * b[1][j];
    } else {
      mulC[j] = 0;
    }
    for (auto k = 1; k < n / 2; ++k) {
      mulC[j] += b[k >> 1][j] * b[(k >> 1) + 1][j];
    }
  }
  for (auto i = 0; i < m; ++i) {
    for (auto j = 0; j < q; ++j) {
      if (n > 1) {
        c[i][j] = (a[i][0] + b[1][j]) * (a[i][1] + b[0][j]) -
            mulR[i] - mulC[j];
      } else {
        c[i][j] = -mulR[i] - mulC[j];
      }
      for (auto k = 1; k < n / 2; ++k) {
        c[i][j] += (a[i][k >> 1] + b[(k >> 1) + 1][j]) * (a[i][(k
            >> 1) + 1] + b[k >> 1][j]);
      }
      if (n % 2 == 1) {
        c[i][j] += a[i][n - 1] * b[n - 1][j];
      }
    }
  }
  return c;
}
```

3.3 Функциональные тесты

В таблице 3.1 представлены функциональные тесты. Для матриц a и b сначала вводятся их размеры. В результате каждый из алгоритмов успешно прошел все тесты.

Таблица 3.1 — Результаты выполнения функциональных тестов

№	Матрица a	Матрица b	Результат (матрица c или ошибка)	Описание теста
1	-1		Ошибка: Некорректный размер	Ошибка в размере
2	2 2 2 2 2 2	3 2	Ошибка: Операция недопустима	Разные размеры
3	1 1 3	1 1 3	9	Минимальный размер
4	2 2 1 2 3 4	2 2 5 6 7 8	19 22 43 50	Квадратные матрицы
5	3 2 1 2 3 4 5 6	2 4 1 2 3 4 5 6 7 8	11 14 17 20 23 30 37 44 35 46 57 68	Разные матрицы

3.4 Оценка ресурсной эффективности

Так как по модели вычислений операции выделения памяти не учитываются, то стоимость строки создания матрицы c принимается за 0. Также не учитывается стоимость возврата значения.

Для стандартного алгоритма перемножения матриц (листинг 3.1) трудоёмкость равна $1+1+m*(1+1+1+1+q*(1+1+3+1+1+n*(1+1+12))) = 2+m*(4+q*(7+14n) = 14mnq+7mq+4m+2$.

Для стандартного алгоритма с оптимизациями (листинг 3.2) трудоёмкость равна $1+1+m*(1+1+1+1+q*(1+1+10+1+1+(n-1)*(1+1+9))) = 2+m*(4+q*(3+11n) = 11mnq+3mq+4m+2$.

Для алгоритма Винограда (листинг 3.3) трудоёмкость равна $1 + 1 + m * (1 + 1 + 1 + 1 + 3 + \frac{n}{2} * (1 + 3 + 15)) + 1 + 1 + q * (1 + 1 + 1 + 1 + 3 + \frac{n}{2} * (1 + 3 + 15)) + 1 + 1 + m * (1 + 1 + 1 + 1 + q * (1 + 1 + 6 + 1 + 3 + \frac{n}{2} * (1 + 3 + 28) + 3 + \begin{cases} 0 & \text{, n четное, лучший случай (л. с.)} \\ 14 & \text{, n нечетное, худший случай (х. с.)} \end{cases})) = 2 + 7m + \frac{19n}{2} + 2 + 7q + \frac{19n}{2} + 2 + m * (4 + q * (12 + 16n + \begin{cases} 0 & \text{, л. с.} \\ 14 & \text{, х. с.} \end{cases})) = 16mnq + \begin{cases} 12 & \text{, л. с.} \\ 26 & \text{, х. с.} \end{cases} mq + \frac{19nq}{2} + \frac{19mn}{2} + 11m + 7q + 6.$

В следующем расчете предполагается, что $n = 1$ (минимальный общий размер у матриц). Это позволяет избавиться от рассмотрения лишних случаев в трёх условных операторах. Для алгоритма Винограда с оптимизациями (листинг 3.4) трудоёмкость равна $1+1+m*(1+1+7+1+3+(\frac{n}{2}-1)*(1+3+11))+1+1+q*(1+1+7+1+3+(\frac{n}{2}-1)*(1+3+11))+1+1+m*(1+1+1+1+q*(1+1+20+1+3+(\frac{n}{2}-1)*(1+3+21)+3+\begin{cases} 0 & \text{, n четное, лучший случай (л. с.)} \\ 11 & \text{, n нечетное, худший случай (х. с.)} \end{cases})) = 2 - 2m + \frac{15n}{2} + 2 - 2q + \frac{15n}{2} + 2 + m * (4 + q * (-13 + \frac{25n}{2} + \begin{cases} 0 & \text{, л. с.} \\ 11 & \text{, х. с.} \end{cases})) = \frac{25mnq}{2} - \begin{cases} 13 & \text{, л. с.} \\ 2 & \text{, х. с.} \end{cases} mq + \frac{15nq}{2} + \frac{15mn}{2} + 2m - 2q + 6.$

Если принять m, n и q равными, то при больших размерах матрицы наименее трудоёмким должен быть оптимизированный стандартный алгоритм, и только затем оптимизированный алгоритм Винограда. Это связано с тем, что в последнем не были решены главные проблемы, такие как деление на 2 на каждой итерации, а также умножение индексов на 2 было заменено на побитовый сдвиг, а не убрано.

3.5 Вывод

В данном разделе был выбран язык программирования для написания программы, были ализованы все ранее описанные алгоритмы, описаны тесты, а также произведена оценка ресурсной эффективности алгоримтов.

4 Исследовательская часть

4.1 Замеры времени работы

Для замеров времени работы функции запускались 10000 раз, на каждой итерации генерировались 2 квадратные матрицы, состоящие из случайных целых чисел в диапазоне от 0 до 1023. Результаты измерений суммировались, после чего выводилось среднее значение. Время работы было замерено с помощью функции $clock\_gettime()$ со значением первого параметра $CLOCK\_PROCESS\_CPUTIME\_ID$ [2]. Все замеры проводились на ноутбуке Acer Swift 3x, процессор 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80ГГц.

Замеры при лучшем случае для алгоритма Винограда (чётное значение n) проводились для матриц размерами 50, 100, 150, 200, 250 строк и столбцов. Результаты представлены на рисунке 4.1.

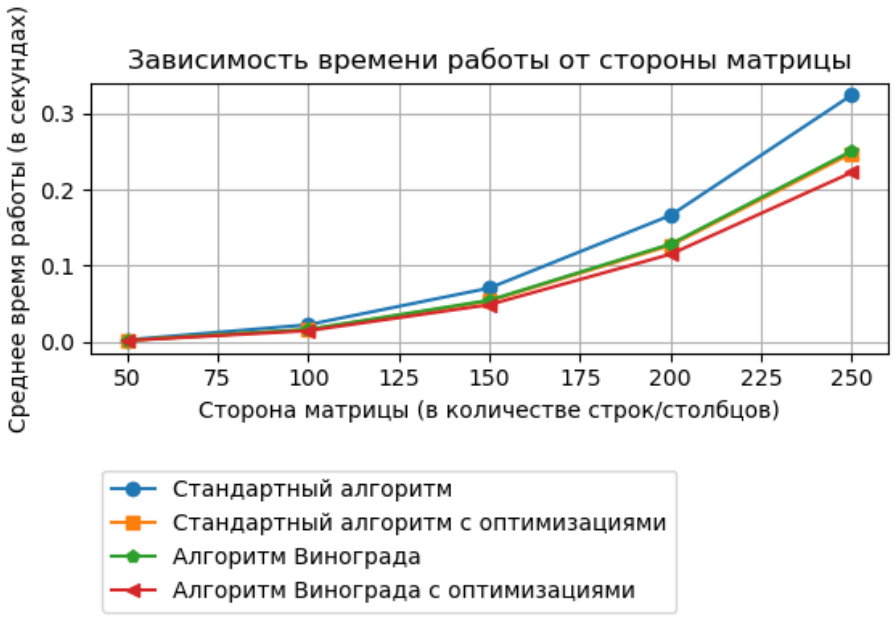

Рисунок 4.1 — Результаты замера времени работы разных алгоритмов (лучший случай)

Результаты не совпали с оценкой ресурсной эффективности, так как реализации алгоритма Винограда оказались быстрее стандартных. Вероятно, это

связано с тем, что умножение куда более трудоёмкая операция, чем было принято в модели вычислений.

Замеры при худшем случае для алгоритма Винограда (нечётное значение n) проводились для матриц размерами 51, 101, 151, 201, 251 строк и столбцов. Результаты представлены на рисунке 4.2

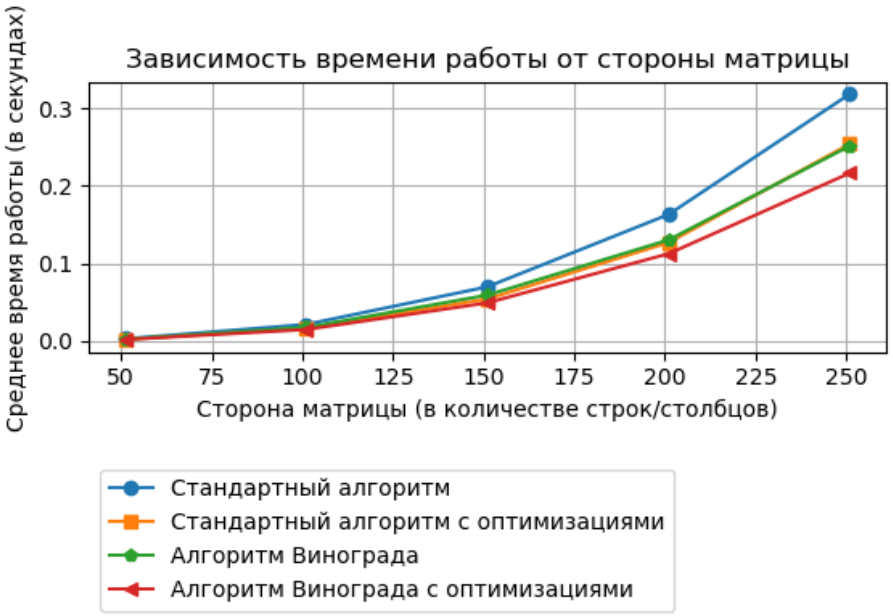

Рисунок 4.2 — Результаты замера времени работы разных алгоритмов (худший случай)

Результаты аналогичны лучшему случаю.

4.2 Вывод

В данном разделе были проведены замеры времени. Самой эффективной оказалась реализация алгоритма Винограда с оптимизациями, наименее эффективной - реализация стандартного алгоритма без оптимизаций.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было выявлено, что алгоритм Винограда оказался эффективнее по времени работы, чем стандартный алгоритм перемножения матриц, хотя подсчитанная по модели вычислений трудоёмкость показала обратное. Оптимизации успешно снизили время работы алгоритмов. Цель работы была достигнута, для чего были описаны стандартный алгоритм и алгоритм Винограда, написана программа, реализующая эти алгоритмы с оптимизациями и без, подсчитана трудоёмкость по описанной модели вычислений, выбраны инструменты для замера процессорного времени, а также проведен анализ затрат реализаций алгоритмов по времени с последующим сравнением с расчётами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Крейнделин В.Б. Григорьева Е. Д. Анализ быстрого алгоритма умножения матриц и векторов для банка цифровых фильтров. T-Comm: Телекоммуникация и транспорт. Том 15. №1, 2021. — С. 4–10.

2. $clock\_gettime(3)$ – Linux manual page [Электронный ресурс]. Режим доступа: `https://man7.org/linux/man-pages/man3/clock_gettime.3.html` (Дата обращения: 15.09.2024).