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 дисциплине "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Петрова А.А.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии.

Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n).

Мы встречаемся с матрицами каждый день, так как любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Примером могут служить:

- список телефонных номеров;
- различные статистические данные;
- табель успеваемости ученика и многое другое.

Целью работы работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда;
- реализовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда;
- дать оценку трудоёмкости алгоритмов;
- замерить время работы алгоритмов;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет проанализирована предметная область и рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда.

Матрица – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Две матрицы одинакового размера можно поэлементно сложить или вычесть друг из 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. При умножении матрицы размером 3x4 на матрицу размером 4x7 мы получаем матрицу размером 3x7. Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга.

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A (1.1) и B (1.2) размерности m на n и n на p соответсвенно:

$$\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \tag{1.1}$$

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,p} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,p} \end{bmatrix} \tag{1.2}$$

В результате получим матрицу C (1.3) размерности m на p:

$$\begin{bmatrix} c_{1,1} & \dots & c_{1,p} \\ \dots & \dots & \dots \\ c_{m,1} & \dots & c_{m,p} \end{bmatrix} \tag{1.3}$$

Формула (1.4) - формула рассчёта элемента, находящегося на i-ой строке j-ого столбца матрицы C [1].

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j} \tag{1.4}$$

1.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее [2].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение (1.5) равно:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.5}$$

Это равенство можно переписать в виде (1.6):

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.6}$$

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. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

1.4 Вычисление трудоёмкости алгоритма

Введем модель трудоемкости для оценки алгоритмов.

- базовые операции стоимостью 1: +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса;
- оценка трудоемкости цикла: f_цикла = f_инициализации + f_сравнения + N * (f_инкремента + f_сравнения + f_тела);
- стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся. В условном операторе может возникнуть лучший и худший случаи по трудоёмкости в зависимости от выполнения условия и в зависимости от входных данных алгоритма.

1.5 Вывод

В данном разделе была проанализирована предметная область и рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда.

Входными данными для программы являются:

- ненулевые размеры первой матрицы;
- количество столбцов второй матрицы (количество строк равно количеству столбцов первой);
- две целочисленные матрицы соответствующих размеров.

Выходные данные: матрица, являющаяся произведением двух заданных матриц.
Ограничения, в рамках которых будет работать программа:

- матрицы непустые;
- заданные матрицы возможно перемножить (матрицы имеют соответствующие размеры);
- элементами матриц являются целые числа;
- корректность данных в пользовательском разделе не проверяется.

Функциональные требования к ПО:

- ПО должно содержать 2 раздела: пользовательский (ручной ввод) и экспериментальный (для замеров времени);
- ПО должно выводить полученную в результате умножения матрицу;
- ПО должно выводить потраченное время.

2 Конструкторская часть

В этом разделе на основе теоретических данных, полученных в аналитическом разделе, будут построены схемы исследуеммых алгоритмов. А также будут описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема классического умножения матриц.

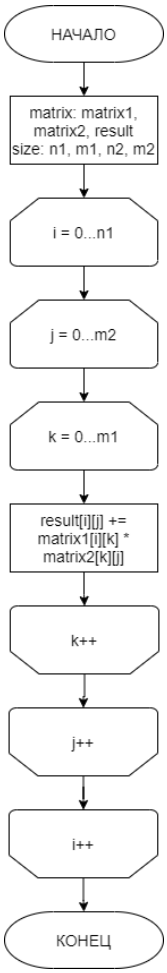

Рис. 2.1: Схема классического алгоритма умножения матриц

На рисунках 2.2, 2.3, 2.4, 2.5 представлена схема алгоритма умножения матриц Винограда.

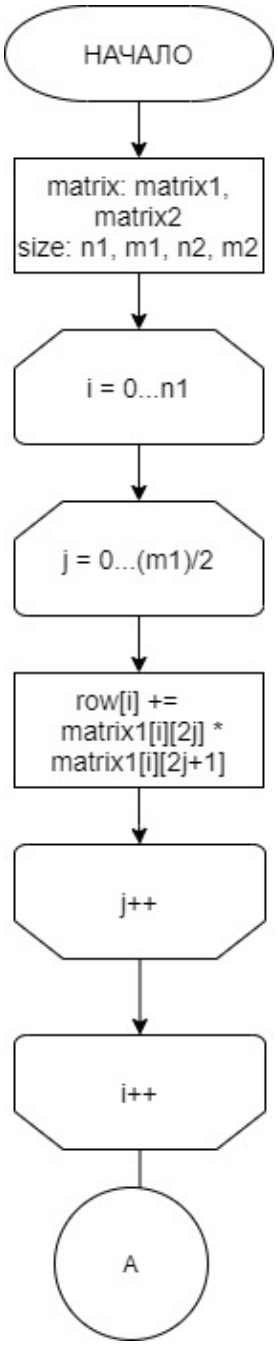

Рис. 2.2: Схема алгоритма Винограда (часть 1)

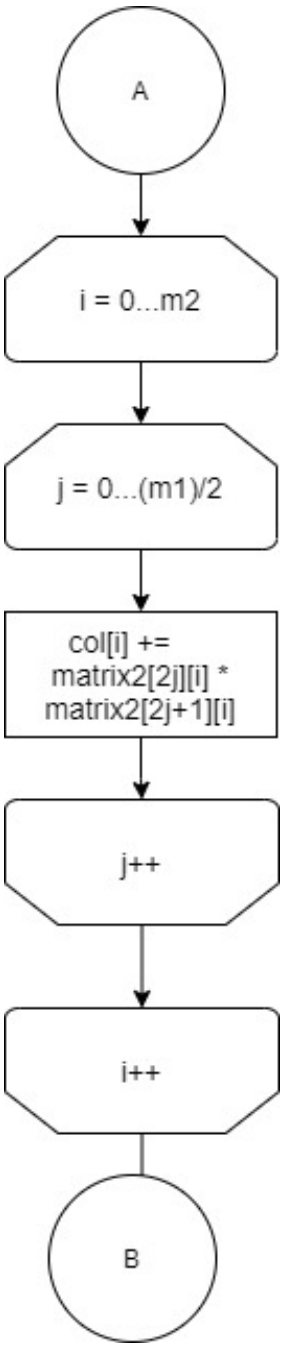

Рис. 2.3: Схема алгоритма Винограда (часть 2)

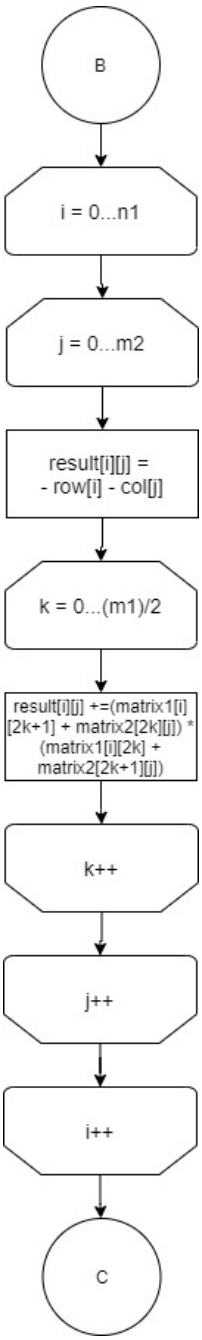

Рис. 2.4: Схема алгоритма Винограда (часть 3)

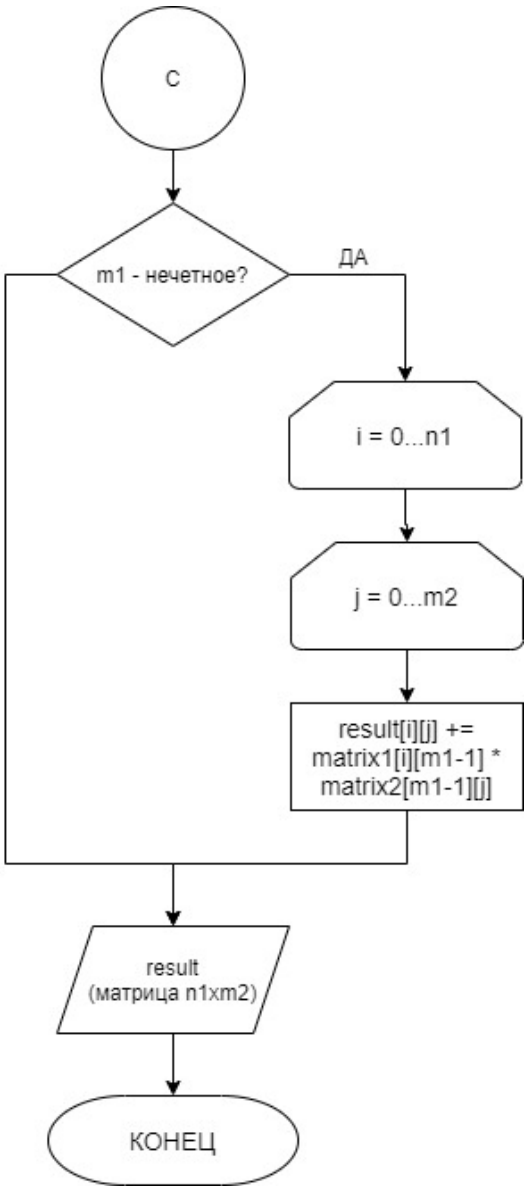

Рис. 2.5: Схема алгоритма Винограда (часть 4)

На рисунках 2.6, 2.7, 2.8, 2.9 представлена схема оптимизированного алгоритма умножения матриц Винограда.

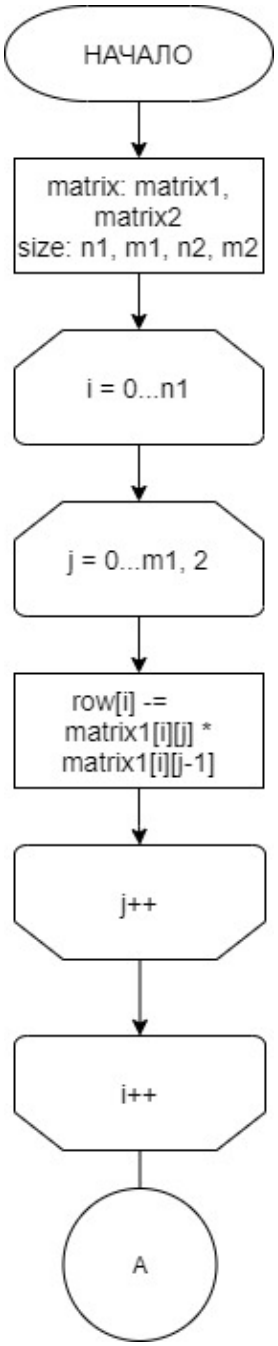

Рис. 2.6: Схема оптимизированного алгоритма Винограда(часть 1)

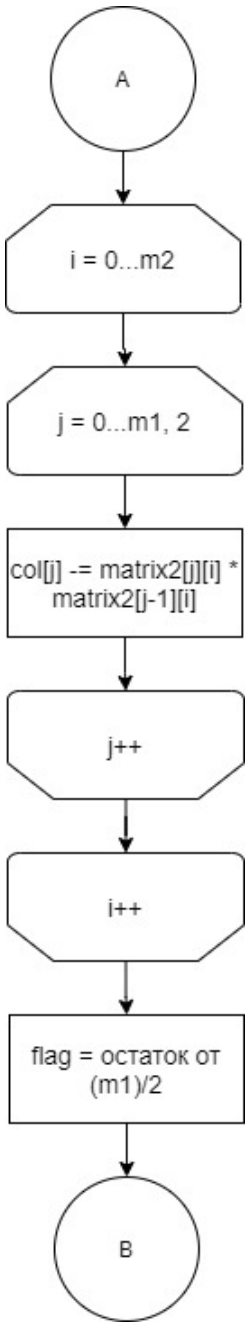

Рис. 2.7: Схема оптимизированного алгоритма Винограда(часть 2)

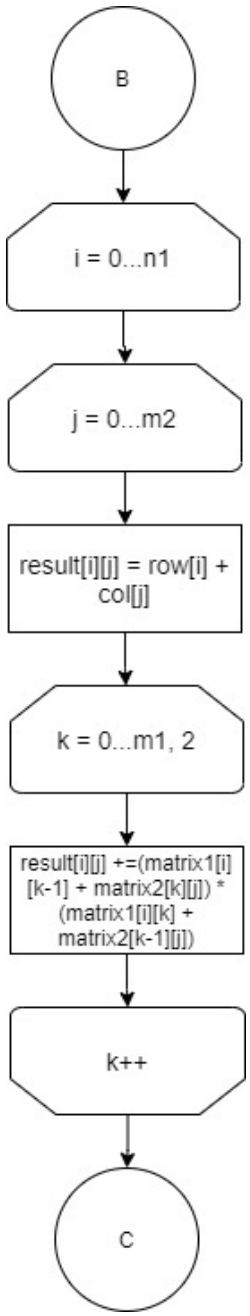

Рис. 2.8: Схема оптимизированного алгоритма Винограда(часть 3)

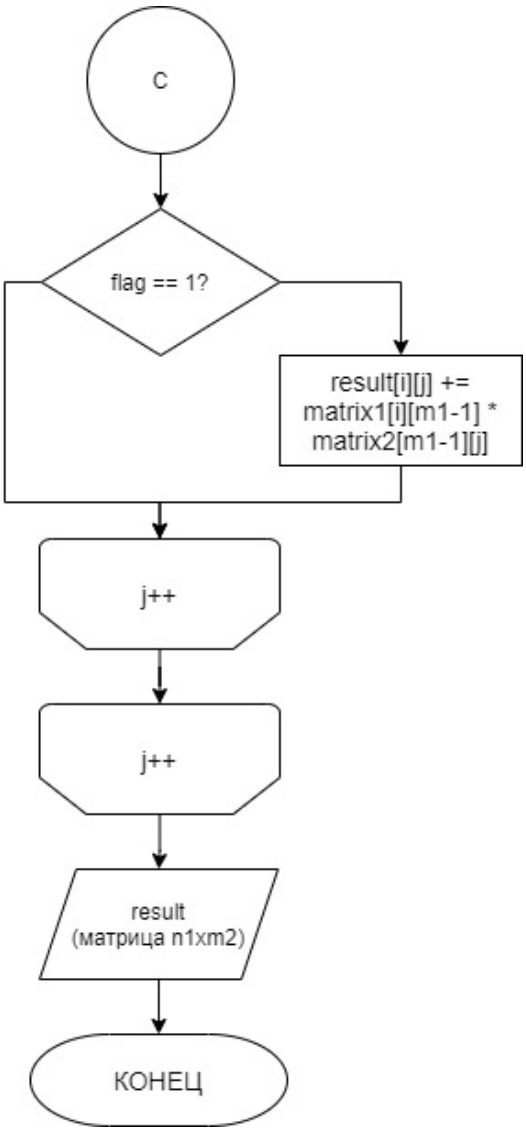

Рис. 2.9: Схема оптимизированного алгоритма Винограда(часть 4)

2.2 Оценка трудоемкости алгоритмов умножения матриц

1. Стандартный алгоритм

$$f = 2 + M(2 + 2 + Q(2 + 2 + N(2 + 8 + 1 + 1 + 1))) = 13\cdot$$

$$\cdot MNQ + 4MQ + 4M + 2 \approx 13 \cdot MNQ$$

2. Алгоритм Винограда

   Трудоемкость алгоритма Винограда:

   Первый цикл: $\frac{15}{2} \cdot NQ + 5 \cdot M + 2$

   Второй цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

   Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, худший случай} \end{bmatrix}$

   Итого: $f = \frac{15}{2}\cdot MN + \frac{15}{2}\cdot QN + 9\cdot M + 8 + 5\cdot Q + 13\cdot MNQ + 12\cdot MQ + \begin{bmatrix} 2 & \text{, в лучшем сл} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, в худшем сл} \end{bmatrix}$

$$f \approx 13 \cdot MNQ$$

3. Оптимизированный алгоритм Винограда

   Введем оптимизации:

   (a) замена операции = на += или -=

   (b) избавление от деления в условиях цикла (j < N, j += 2)

   (c) Заносим проверку на нечетность кол-ва строк внутрь основных циклов

   (d) Расчет условия для последнего цикла один раз, а далее использование флага

   Первый цикл: $4 \cdot NQ + 4 \cdot M + 2$

   Второй цикл: $4 \cdot MN + 4 \cdot M + 2$

   Третий цикл: $9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 10 \cdot QM & \text{, худший случай} \end{bmatrix}$

Трудоемкость оптимизированного алгоритма Винограда:

Итого:

$$f = 4 \cdot NQ + 4 \cdot M + 2 + 4 \cdot MN + 4 \cdot M + 2 + 9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2 +$$

$$+ \begin{bmatrix} 2 & ,\text{л.с} \\ 10 \cdot QM & ,\text{х.с} \end{bmatrix} \approx 9 \cdot MNQ$$

2.3 Структуры данных

Для хранения матриц будут использоваться двумерные массивы. Для хранения предварительных вычислений в алгоритме Винограда - одномерные массивы.

2.4 Тестирование и классы эквивалентности

Для проверки работоспособности ПО будет применяться функциональное тестирование. Классы эквивалентности:

- квадратные матрицы одинаковых размеров;
- матрицы разных размеров (подходящие для умножения);
- в матрицах присутствуют как положительные, так и отрицательные числа;
- матрицы из одного элемента.

2.5 Используемая память

Все используемые в данной работе массивы будут динамическими. В связи с этим, нужно будет контролировать проблемы с памятью, в частности при выделении памяти под массивы и её освобождении из-под них.

Из всего вышесказанного следует, что количество памяти, необходимой алгоритмам, пропорционально размерам матриц.

2.6 Структура ПО

ПО будет состоять из набора функций:

- основная функция, работающая с меню;
- ввод исходных данных с клавиатуры;
- классический алгоритм умножения матриц;
- алгоритм Винограда;

- оптимизированный алгоритм Винограда;
- замеры времени.

2.7 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы иследуеммых алгоритмов. Также были описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

3 Технологическая часть

В данном разделе будут разработаны исходные коды трех алгоритмов: классическое умножение матриц, алгоритм Винограда, оптимизированный алгоритм Винограда.

3.1 Средства реализации

Для реализации программы умножения двух матриц был выбран язык программирования Python. Данный выбор обусловлен тем, что в этом языке присутсвует функция для измерения процессорного времени.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация алгоритмов умножения: стандартного, Винограда и Винограда с оптимизацией.

Листинг 3.1: Стандартный алгоритм умножения

```
1  def std_mult(matr_a, matr_b, M, N, Q):
2    res = [[0 for x in range(Q)] for y in range(M)]
3      for i in range(M):
4        for j in range(Q):
5          for k in range(N):
6            res[i][j] = res[i][j] + matr_a[i][k] * matr_b[k][j]
7    return res
```

Листинг 3.2: Алгоритм Винограда

```
  def precomp_rows(matr, M, N):
    mul_h = [0 for x in range(M)]
      for i in range(M):
        for j in range(0, N - 1, 2):
          mul_h[i] = mul_h[i] + matr[i][j] * matr[i][j + 1]
    return mul_h

  def precomp_cols(matr, M, N):
    mul_v = [0 for y in range(N)]
    for i in range(N):
      for j in range(0, M - 1, 2):
        mul_v[i] = mul_v[i] + matr[j][i] * matr[j + 1][i]
    return mul_v

  def win_mult(matr_a, matr_b, M, N, Q):
    res = [[0 for x in range(Q)] for y in range(M)]

    mul_h = precomp_rows(matr_a, M, N)
    mul_v = precomp_cols(matr_b, N, Q)

    for i in range(M):
      for j in range(Q):
        res[i][j] = -mul_h[i] - mul_v[j]
        for k in range(0, N - 1 , 2):
          res[i][j] = res[i][j] + (matr_a[i][k] + matr_b[k + 1][j]) * (
              matr_a[i][k + 1] + matr_b[k][j])
    if N % 2 != 0:
      for i in range(M):
        for j in range(Q):
          res[i][j] = res[i][j] + matr_a[i][N - 1] * matr_b[N - 1][j]
    return res
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
def opt_precomp_rows(matr, M, N):
  mul_h = [0 for x in range(M)]
  for i in range(M):
    for j in range(0, N - 1, 2):
      mul_h[i] += (matr[i][j] * matr[i][j + 1])
  return mul_h

def opt_precomp_cols(matr, M, N):
  mul_v = [0 for y in range(N)]
  for i in range(N):
    for j in range(0, M - 1, 2):
      mul_v[i] += (matr[j][i] * matr[j + 1][i])
  return mul_v

def opt_win_mult(matr_a, matr_b, M, N, Q):
  res = [[0 for x in range(Q)] for y in range(M)]

  mul_h = opt_precomp_rows(matr_a, M, N)
  mul_v = opt_precomp_cols(matr_b, N, Q)

  odd = N % 2

  for i in range(M):
    for j in range(Q):
      res[i][j] = -mul_h[i] - mul_v[j]
      for k in range(0, N - 1 , 2):
        res[i][j] += ((matr_a[i][k] + matr_b[k + 1][j]) * (matr_a[i][k +
            1] + matr_b[k][j]))
      if odd:
        res[i][j] += (matr_a[i][N - 1] * matr_b[N - 1][j])
  return res
```

3.3 Тестовые данные

В таблице 3.1 приведены тестовые данные, на которых было протестированно разработанное ПО. Как видно из этой таблицы, все тесты были успешно пройдены, что означает, что программа работает правильно.

Первая матрица	Вторая матрица	Ожидаемый результат	Полученный результат
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$
$\begin{bmatrix} 8 & -10 & 4 \\ 0 & -7 & 6 \end{bmatrix}$	$\begin{bmatrix} -7 & 1 \\ -9 & 0 \\ 7 & -3 \end{bmatrix}$	$\begin{bmatrix} 62 & -4 \\ 105 & -18 \end{bmatrix}$	$\begin{bmatrix} 62 & -4 \\ 105 & -18 \end{bmatrix}$
$\begin{bmatrix} 10 \end{bmatrix}$	$\begin{bmatrix} -5 \end{bmatrix}$	$\begin{bmatrix} -50 \end{bmatrix}$	$\begin{bmatrix} -50 \end{bmatrix}$

3.4 Вывод

В данном разделе были разработаны исходные коды трех алгоритмов: классическое умножение матриц, алгоритм Винограда, оптимизированный алгоритм Винограда, основная разница которого — наличие предварительной обработки, а также уменьшение количества операций умножения для увеличения эффективности.

4 Исследовательская часть

В этом разделе будет проведён эксперимент для определения эффективности по времени работы каждого из разработанных алгоритмов.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Меню:
1. Ручной ввод
2. Замеры процессорного времени выполнения алгоритмов
0. Выход

Выберите пункт меню: 1

Матрица A:
Введите кол-во строк: 2
Введите кол-во столбцов: 3
Введите матрицу:
8
-10
4
0
-7
6

Матрица B:
Введите кол-во столбцов: 2
Введите матрицу:
-7
1
-9
0
7
-3

Обычное умножение:
62 -4
105 -18
Алгоритм Винограда:
62 -4
105 -18
Улучшенный алгоритм Винограда:
62 -4
105 -18
```

Рис. 4.1: Работа алгоритмов нахождения расстояния Левенштейна и Дамерау – Левенштейна.

4.2 Технические характеристики

Ниже приведеные технические характеристики устройства, на котором было проведенно тестирование ПО:

- Операционная система: Windows 10 64-bit Home [3].
- Оперативная память: 8 GB.
- Процессор: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz [4].

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью функции process_time модуля time в Python [5]. Данная функция возвращает значение в долях секунды суммы системного и пользовательского процессорного времени текущего процесса.

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

Таблица 4.1: Лучший случай времени выполнения алгоритмов

Размер матриц	Стандартный	Виноград	Оптимизированный
50	0.019375	0.017344	0.017344
100	0.133750	0.125625	0.122813
150	0.452813	0.411406	0.400313
200	1.034063	0.937344	0.926719
250	1.989219	1.825156	1.795313

Таблица 4.2: Худший случай времени выполнения алгоритмов

Размер матриц	Стандартный	Виноград	Оптимизированный
51	0.019062	0.018594	0.018281
101	0.137031	0.127812	0.126719
151	0.445937	0.410156	0.406875
201	1.045000	0.958750	0.949375
251	2.020000	1.839063	1.825469

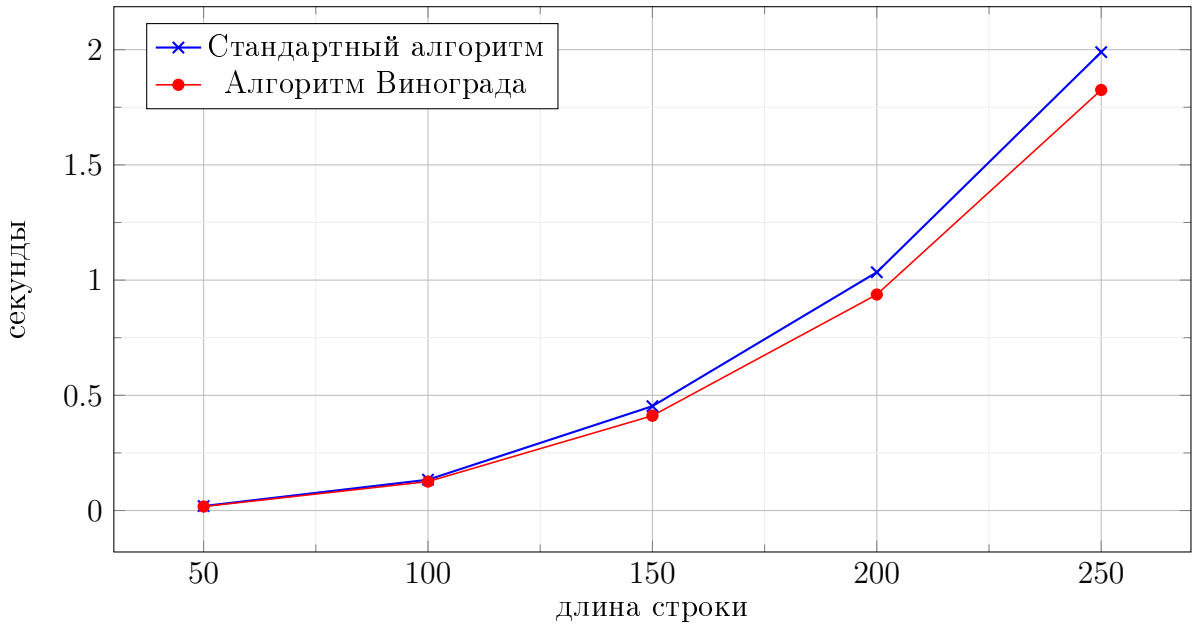

Рис. 4.2: Сравнение стандартного алгоритма умножения и алгоритма Винограда

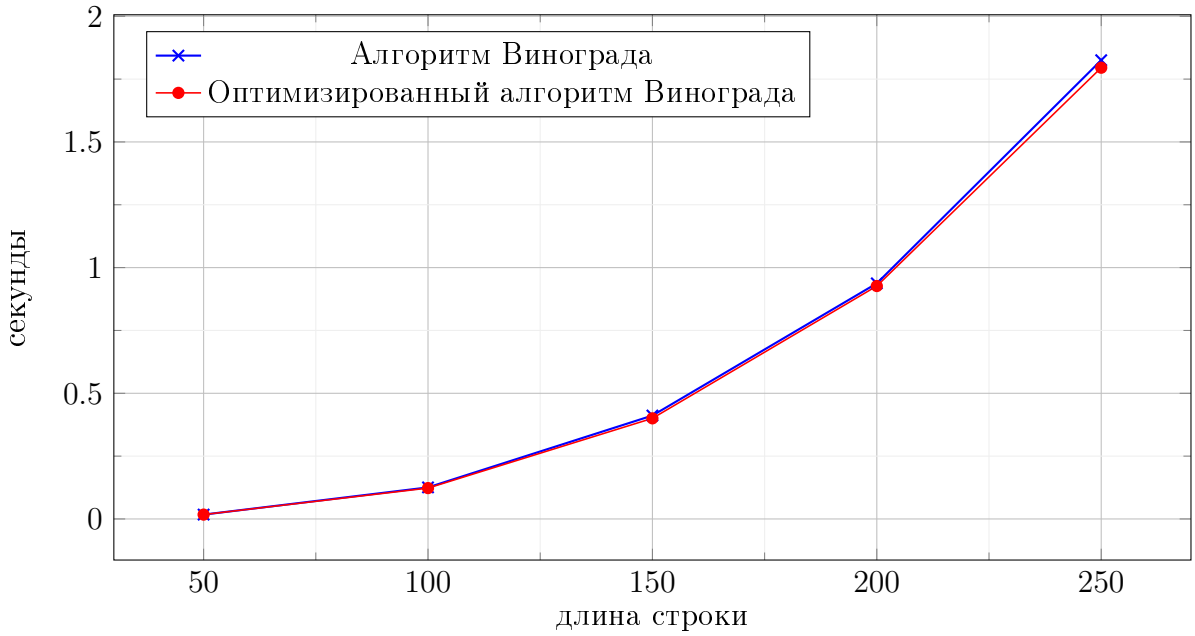

Рис. 4.3: Сравнение алгоритма Винограда и оптимизированного алгоритма Винограда

4.4 Вывод

В результате проведенного эксперимента можно сделать вывод, что алгоритм Винограда эффективнее стандартного алгоритма по времени в среднем примерно на 8,7%.

Заключение

В ходе проделанной работы были изучены и реализованы классический алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

Экспериментально было подтверждено различие по временной эффективности алгоритмов умножения матриц на материале замеров процессорного времени выполнения реализации на варьирующихся размерах матриц. Так, оптимизированный алгоритм Винограда работает чуть быстрее обычного алгоритма Винограда и зачительно быстрее классического алгоритма умножения матриц.

На основании сравнения данных алгоритмов был сделан вывод, что алгоритм Винограда эффективнее классического алгоритма в среднем примерно на 8,7%. При этом оптимизированный алгоритм Винограда сравним по времени с обычным алгоритмом Винограда.

Литература

[1] Алгоритм умножения матриц. Режим доступа: https://www.math10.com/ru/vysshaya-matematika/matrix/umnozhenie-matric.html. Дата обращения: 20.10.2021.

[2] Умножение матриц по Винограду. Режим доступа: http://algolib.narod.ru/Math/Matrix.html. Дата обращения: 20.10.2021.

[3] Windows 10 Pro и Windows 10 Домашняя. Режим доступа: https://www.microsoft.com/ru-ru/windows/compare-windows-10-home-vs-pro. Дата обращения: 18.10.2021.

[4] Процессор Intel® Core™ i3-1115G4 (6 МБ кэш-памяти, до 4,10 ГГц). Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208652/intel-core-i31115g4-processor-6m-cache-up-to-4-10-ghz/specifications.html. Дата обращения: 14.10.2021.

[5] Функция process_time() модуля time в Python. Режим доступа: https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time/. Дата обращения: 5.10.2021.