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

Студент Топорков Павел
Группа ИУ7-53Б
Дисциплина Анализ алгоритмов

Преподаватели: _____ Строганов Ю.В., Волкова Л.Л.
подпись, дата Фамилия, И.О.

Оценка _____

Москва — 2021 г.

Оглавление

Введение

Цель работы: изучение алгоритмов умножения матриц в данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Следует изучить и имплементировать указанные алгоритмы. Помимо этого требуется произвести расчет сложности каждого алгоритма, получить практические навыки в улучшении алгоритмов.

В ходе лабораторной работы предстоит:

1. изучить два алгоритма умножения матриц: стандартный и алгоритм Винограда;
2. оптимизировать алгоритм Винограда;
3. дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;
4. имплементировать три алгоритма умножения матриц на одном из языков программирования;
5. сравнить алгоритмы умножения матриц.

1 Аналитическая часть

1.1 Стандартный алгоритм

Матрицей A размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. [1] Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно матрично перемножить. У произведения будет столько же строк, сколько в первой матрице, и такое же число столбцов, сколько во второй.

Рассмотрим на примере: пусть даны две прямоугольные матрицы A и B размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

Следующая формула выражает произведение матриц:
$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$. [1]

1.2 Алгоритм Винограда

Подход Алгоритма Винограда (англ. the Winograd algorithm) является иллюстрацией общей методологии, начатой в 1979-х годах на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены. [2]

Рассмотрим два вектора $X = (x1, x2, x3, x4)$ и $Y = (y1, y2, y3, y4)$.

Их скалярное произведение равно (1.1)

$$X \cdot Y = x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + x_4 \cdot y_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде (1.2)

$$X \cdot Y = (x_1 + y_2) \cdot (x_2 + y_1) + (x_3 + y_4) \cdot (x_4 + y_3) - x_1 \cdot x_2 - x_3 \cdot x_4 - y_1 \cdot y_2 - y_3 \cdot y_4 \tag{1.2}$$

Можно заметить, что выражение в правой части последнего равенства допускает некоторую предварительную обработку: его части мож-

но 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, что вычислительно менее затратно, чем «классическое» умножение матриц.

Вывод

В этом разделе были рассмотрены алгоритмы «классического» умножения матриц и алгоритм Винограда. Основное отличие этих алгоритмов друг от друга — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

Требования к вводу программы:

- На вход подаются две матрицы.

Требования к программе:

- Корректное умножение двух матриц.
- Если введены неподходящих размеров матрицы, программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

Далее будут приведены схемы алгоритмов «классического» умножения матриц (рис. 2.1), алгоритма Винограда (рис. 2.2) и оптимизированного алгоритма Винограда (рис. 2.3).

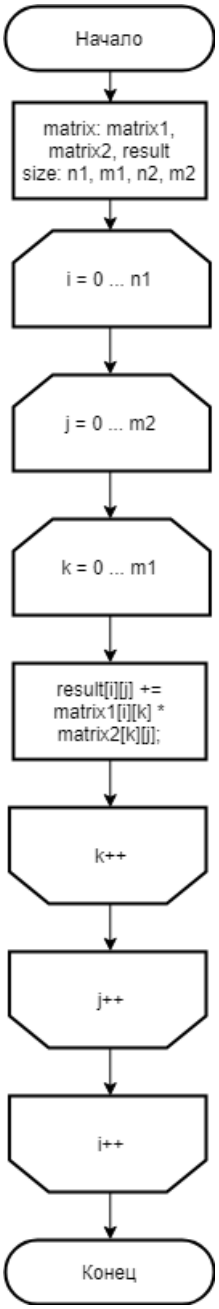

Рисунок 2.1: Схема «классического» алгоритма умножения матриц

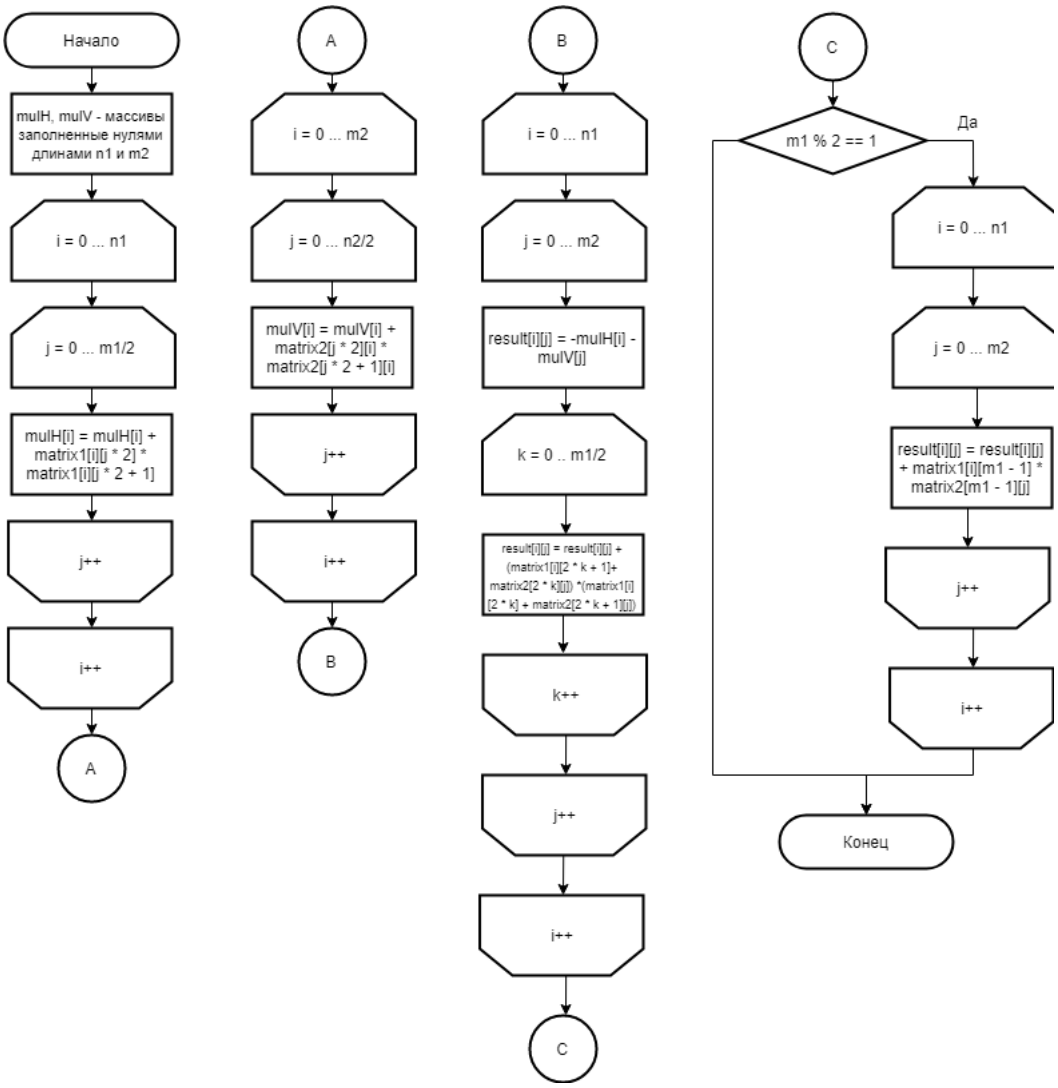

Рисунок 2.2: Схема алгоритма Винограда

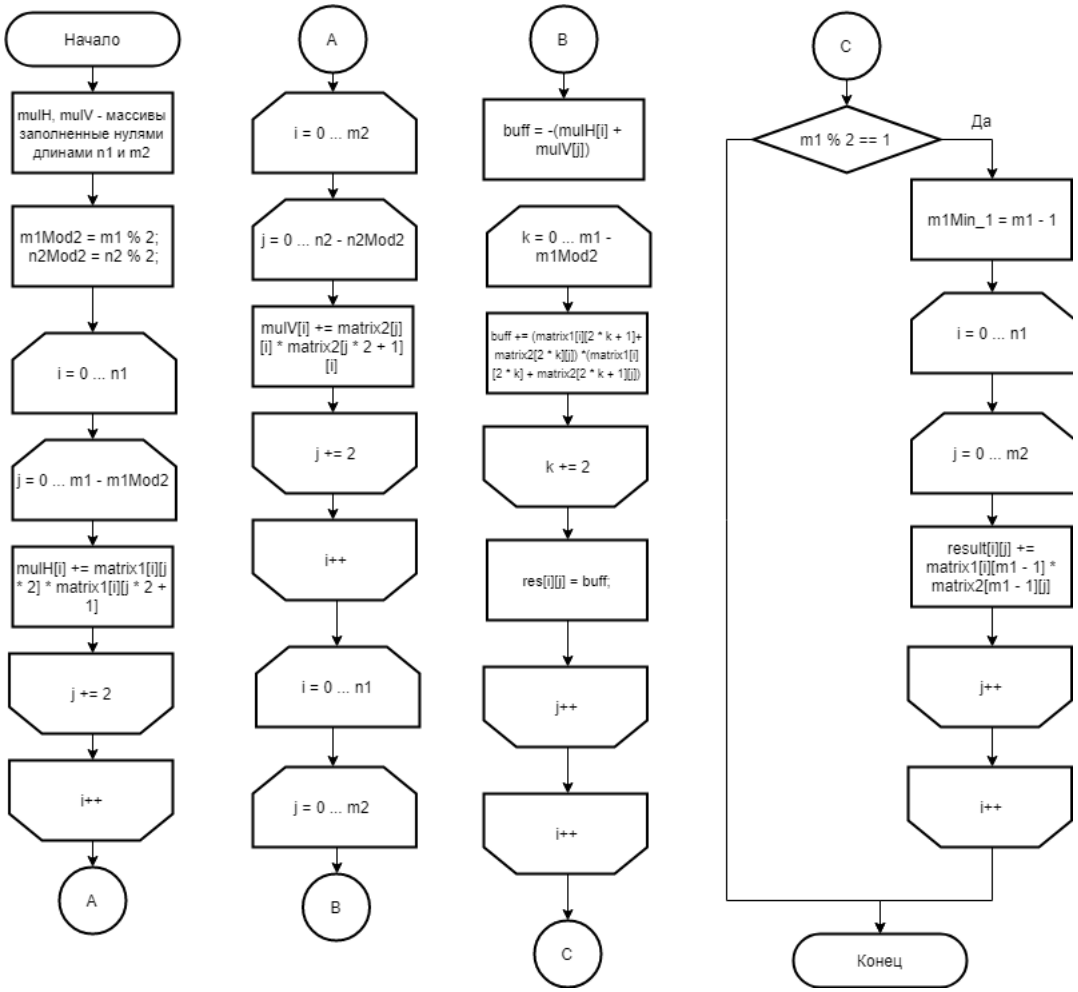

Рисунок 2.3: Схема оптимизированного алгоритма Винограда

Вывод

В данном разделе были рассмотрены требования для программы, а также схемы алгоритмов умножения матриц.

3 Технологическая часть

3.1 Выбор ЯП

Для реализации программы был выбран язык программирования Python 3 в следствие относительной простоты написание кода на данном ЯП. [3] Среда разработки - редактор кода VS Code. [4]

3.2 Листинги кода алгоритмов

3.2.1 Стандартный алгоритм перемножения и алгоритм Винограда

Далее будут приведены листинги алгоритмов «классического» умножения матриц (листинг. 3.1) и алгоритма Винограда (листинг 3.2).

Листинг 3.1: Стандартный алгоритм умножения матриц

```
def dot_product(first, second, i, j):
  product = 0
  for idx in range(len(first[0])):
    product += first[i][idx] * second[idx][j]
  return product

def trivial_m12n(a, b):
  result = [[0 for _ in range(len(b[0]))] for _ in range(len(a))]
  for i in range(len(a)):
    for j in range(len(b[0])):
      result[i][j] = dot_product(a, b, i, j)
  return result
```

Листинг 3.2: Алгоритм Винограда

```
def winograd_m12n(a, b):
  row_factors = []
  col_factors = []
  result = [[0 for _ in range(len(b[0]))] for _ in range(len(a))]

  for i in range(len(a)):
    value = 0
      for j in range(len(a[0]) // 2):
        value = value + a[i][j * 2] * a[i][j * 2 + 1]
    row_factors.append(value)

  for i in range(len(b[0])):
    value = 0
```

```
    for j in range(len(b) // 2):
      value = value + b[j * 2][i] * b[j * 2 + 1][i]
    col_factors.append(value)

  for i in range(len(a)):
    for j in range(len(b[0])):
      result[i][j] = -row_factors[i] - col_factors[j]

  for idx in range(len(a[0]) // 2):
    result[i][j] = (a[i][2 * idx + 1] + b[2 * idx][j]) * (a[i][2 * idx] + b[2 * idx + 1][j]) +
        result[i][j]

  if len(a) % 2:
    for i in range(len(a)):
      for j in range(len(b[0])):
        result[i][j] = result[i][j] + a[i][len(a[0]) - 1] * b[len(a[0])- 1][j]

  return result
```

3.2.2 Оптимизированный алгоритм Винограда

В рамках данной лабораторной работы мною было предложено 3 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена (value = value + . . .) на (value +=);
3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти и сброс буфера в ячейку матрицы после цикла.

Ниже приведен листинг оптимизированного алгоритма Винограда (листинг 3.3).

Листинг 3.3: Оптимизированный алгоритм Винограда

```
def better_winograd_m12n(a, b):
  row_factors = []
  col_factors = []
  rows_a = len(a); cols_a = len(a[0]); cols_amod = cols_a // 2
  rows_b = len(b); cols_b = len(b[0]); rows_bmod = rows_b // 2
  result = [[0 for _ in range(cols_b)] for _ in range(rows_a)]

  for i in range(rows_a):
    value = 0
    for j in range(cols_amod):
      value += a[i][j * 2] * a[i][j * 2 + 1]
    row_factors.append(value)

```

```
    for i in range(cols_b):
      value = 0
      for j in range(rows_bmod):
        value = value + b[j * 2][i] * b[j * 2 + 1][i]
      col_factors.append(value)

    for i in range(rows_a):
      for j in range(cols_b):
        buffer = -row_factors[i] - col_factors[j]
        for idx in range(cols_amod):
          buffer += (a[i][2 * idx + 1] + b[2 * idx][j]) * (a[i][2 * idx] + b[2 * idx + 1][j])
        result[i][j] = buffer

    if len(a) % 2:
      for i in range(rows_a):
        for j in range(cols_b):
          result[i][j] += a[i][cols_a - 1] * b[cols_a - 1][j]

    return result
```

3.3 Тесты

Проведем тестирование программы в таблице 3.1. В столбцах "Множитель №1"и "Множитель №2"перемножаемые матрицы.

Таблица 3.1: Таблица тестовых данных

№	Множитель №1	Множитель №2	Ожидаемый рез.	Полученный рез.
1	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	error	error
2	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{pmatrix}$	$\begin{pmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{pmatrix}$
3	$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}4\\5\\6\end{pmatrix}$	$\begin{pmatrix}32\end{pmatrix}$	$\begin{pmatrix}32\end{pmatrix}$
4	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$

Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы; также убедились в корректности работы разработанного ПО.

4 Исследовательская часть

4.1 Сравнение алгоритмов

Был проведен замер времени работы каждого из алгоритмов на машине HP ProBook 450 G5. [5] Для минимизации погрешности замеров времени каждый алгоритм исполнялся над одними и теми же строками 100 раз и затраченное время делилось на 100, для получения усредненного времени выполнения. Отмечу, что замеры ремени работы алгоритмов на матрицах размером 500x500 проводился единожды, поскольку алгоритмы исполнялись относительно долго.

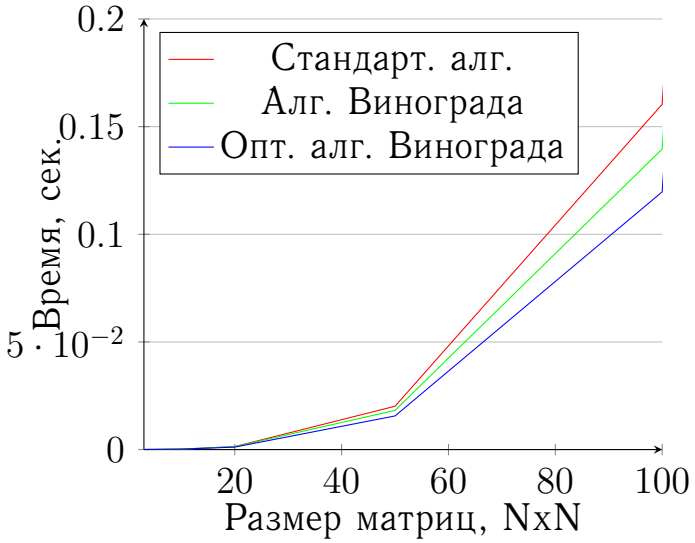

Таблица 4.1: Процессорное время исполнения (в секундах)

Размер	Стандартный алг.	Алг. Винограда	Опт. алг. Винограда
3	0.00005897	0.00006329	0.00005996
5	0.00009348	0.00012000	0.00008122
10	0.00019870	0.00025753	0.00020099
20	0.00138449	0.00133634	0.00114747
50	0.02012375	0.01821243	0.01562920
100	0.16036983	0.13958475	0.11971728
500	25.17392058	22.15858002	20.09244162

Вывод

Самый быстродейственным алгоритмом является оптимизированный алгоритм Винограда: на каждом размере матриц он выполнялся быстрее двух других алгоритмов. «Классический» алгоритм перемножения матриц хорошо себя показывал на матрицах относительно маленького размера, однако уже для матриц, размерностями больше, чем 50x50, он существенно уступал алгоритму Винограда. В среднем улучшенный алгоритм Винограда выполняется быстрее «обычного» алгоритма Винограда на 15%.

Заключение

В ходе лабораторной работы были изучен алгоритм Винограда для перемножения матриц, получены практические навыки реализации указанного алгоритма. Также была произведена оптимизация данного алгоритма для ПО.

Экспериментально было подтверждено различие по временным затратам классического алгоритма перемножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализованных алгоритмов на матрицах различной размерности.

Было установлено, что самым быстродейственным алгоритмом является оптимизированный алгоритм Винограда. «Классический» алгоритм перемножения матриц хорошо себя показывал на матрицах относительно маленького размера, однако существенно уступал в производительностью алгоритму Винограда на относительно больших матрицах. Также было экспериментально выявлено, что улучшенный алгоритм Винограда выполняется быстрее «обычного» алгоритма Винограда в среднем на 13-17%.

Подытожив, оптимизированный алгоритм Винограда заметно выигрывает у стандартного алгоритма на матрицах больших размерностей, что делает его более применим в задачах, где работают с относительно большими матрицами.

Литература

[1] И.В. Белоусов. Матрицы и определители, учебное пособие по линейной алгебре. Интститут прикладной физики, 2006. С. 1–16.

[2] Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 1990. P. 9:251–280.

[3] Язык программирования Python. Режим доступа: `http://www.python.org` (дата обращения: 01.10.2020).

[4] Редактор кода VS Code. Режим доступа: `https://code.visualstudio.com/` (дата обращения: 01.10.2020).

[5] Технические характеристики ноутбука HP ProBook 450 G5 PC. Режим доступа: `https://support.hp.com/ru-ru/document/c05739572` (дата обращения: 01.10.2020).