*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №2**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Умножение матриц**

Студент гр. ИУ7-51Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Громова В.П.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[**Введение**](#_k5dnresp2ym9) **3**

[**Аналитическая часть**](#_8y2m4p612xwl) **4**

[Описание алгоритмов](#_hxixtc4k4y3r) 4

[Описание модели вычислений](#_l0mwksllceyh) 5

[**Конструкторская часть**](#_e605ypxc4hq) **6**

[Схемы алгоритмов](#_5u6pxkyjbb59) 6

[Оптимизация алгоритма Винограда](#_qynd0csivo66) 11

[Вычисление трудоемкости алгоритмов](#_hqa7ecywmdob) 11

[**Технологическая часть**](#_te5b7ppyrcfk) **13**

[Требования к ПО](#_dflahe5brn4y) 13

[Средства реализации](#_tzfvlwapbdm8) 13

[Листинги кода](#_a7ffra2y4hml) 13

[**Экспериментальная часть**](#_hafwjwrs4nle) **15**

[Примеры работы программы](#_8dhvbm9mn6x2) 15

[Результаты функционального тестирования](#_nqqpoo6pesa6) 16

[Постановка эксперимента по замеру времени](#_qskq9s8tr9ha) 17

[Сравнительный анализ на материале экспериментальных данных](#_osqjva9o5i2k) 17

[**Заключение**](#_g6gpwoin67m5) **20**

### Введение

Цель данной лабораторной работы: изучить алгоритмы умножения матриц и исследовать их трудоемкость.

Задачи лабораторной работы:

1. Реализовать стандартный алгоритм умножения матриц.
2. Реализовать алгоритм Винограда умножения матриц.
3. Оптимизировать алгоритм Винограда.
4. Рассчитать трудоемкость алгоритмов.
5. Сравнить реализованные алгоритмы по времени.
6. Описать проделанную работу и обосновать получившиеся результаты.

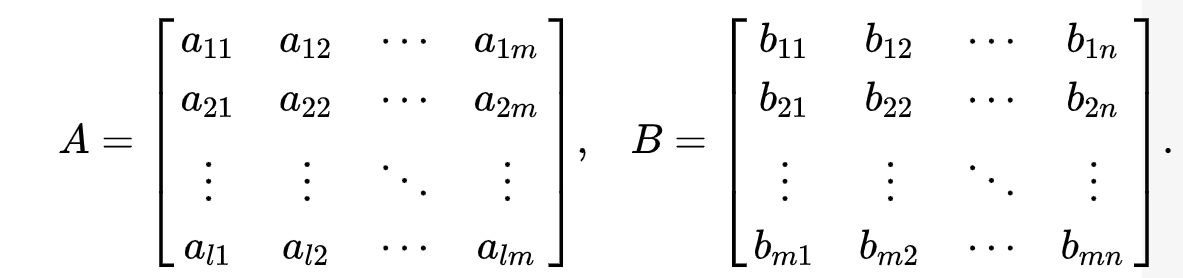
### Аналитическая часть

#### Описание алгоритмов

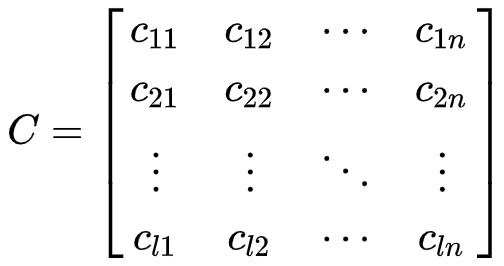
* + 1. Стандартный алгоритм умножения матриц

Для вычисления произведения двух матриц A и B каждая строка матрицы A умножается на каждый столбец матрицы B. Затем сумма данных произведений записывается в соответствующую ячейку результирующей матрицы C.

Пусть даны две прямоугольные матрицы A и B размерности l x m и m x n соответственно:



Тогда матрица C размерностью l x n:

,

где

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.

* + 1. Алгоритм Винограда умножения матриц

Алгоритм Винограда основан на сокращении количества операций умножения в алгоритме для повышения его эффективности. Результат умножения двух матриц представляет собой скалярное произведение соответствующего строки и столбца. 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 • W = v1w1 + v2w2 + v3w3 + v4w4.

Это равенство можно переписать в виде: V • W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) — v1v2 — v3v4 — w1w2 — w3w4.

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: вместо четырех умножений - шесть, а вместо трех сложений - десять, 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, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. В случае нечетного количества столбцов первой матрицы, необходимо будет провести дополнительные вычисления: к каждой ячейки результирующей матрицы Cij прибавить произведение Ai(n-1) \* B(n-1)j, где n - количество столбцов в первой матрице ( и строк -- во второй).

#### Описание модели вычислений

Для того, чтобы рассчитать трудоемкость алгоритмов, будет использована следующая модель вычислений:

* трудоемкость базовых операций +, -,, =, <, >, <=, >=, ==, !=, [], +=, -= будет оцениваться в единицу, операции \*, /, %, //(целочисленное деление) оцениваются в 2;
* трудоемкость условного перехода будет оцениваться в ноль;
* трудоемкость цикла будет оцениваться по следующей формуле:

fцикла = fинициализации + fсравнения + N\*(f тела цикла + f инкремента + f сравнения), где N - количество повторений тела цикла.

### Конструкторская часть

#### Схемы алгоритмов

На рисунках 2.1 - 2.5 представлены схемы алгоритмов умножения входной матрицы mtr1 размером q на m на входную матрицу mtr2 размером m на n.

|  |
| --- |
|  |
| *рис. 2.1 - стандартный алгоритм умножения матриц* |

|  |  |
| --- | --- |
|  |  |

*рис. 2.2 - алгоритм Винограда умножения матриц*

|  |
| --- |
|  |
| *рис. 2.3 - алгоритм Винограда умножения матриц (продолжение)* |

|  |  |
| --- | --- |
|  |  |

*рис. 2.4 - оптимизированный алгоритм Винограда*

|  |
| --- |
|  |
| *рис. 2.5 - оптимизированный алгоритм Винограда (продолжение)* |

#### Оптимизация алгоритма Винограда

Для того, чтобы оптимизировать алгоритм Винограда, были внесены следующие изменения:

* были заранее вычислены значения d = m -1 и m2 = m % 2, чтобы не вычислять их в цикле на каждой итерации;
* в циклах для подсчета значений дополнительных векторов вложенный цикл по k от 0 до m//2 был заменен на цикл по k от 0 до (m - 1) с шагом 2. Так удалось избежать одной операции деления и двух операций умножений на каждой итерации внутреннего цикла. При этом значения в дополнительных векторах учитывались со знаком ‘-’ для того, чтобы далее в основном цикле избежать двух операций ‘-’ на каждой итерации внутреннего цикла;
* в основном цикле вложенный цикл по k также был изменен (см. предыдущий пункт). Также была добавлена буферная переменная buf, чтобы снизить количество обращений к памяти (операция []);
* проверка на нечетное количество столбцов в первой матрице была внесена во внутренний цикл по j. Это ухудшило значение трудоемкости в лучшем случае. Однако значительно улучшило это значение при худшем случае (не нужно еще раз “проходить” всю результирующую матрицу для добавления одного значения).

#### Вычисление трудоемкости алгоритмов

Вычисление трудоемкости стандартного алгоритм умножения матриц:

fa = 2 + q \* (2 + 2 + n \* (2 + 2 + m \* (2 + 8 + 1 + 1 + 2) ) ) = 2 + q \* (4 + n \* (4 + 13 \*m) ) = 14 \* q \* n \* m + 4 \* q \* n + 4 \* q + 2 ≈ **14 \* q \* n \* m**

Вычисление трудоемкости алгоритма Винограда:

fa = fr + fc + fb + fo = 16 \* q \* n \* m + 29 \* q \* n + (19/2) \* n \*m + (19/2) \* q \* m + 14 \* q + 6 \* n + 11 (в худшем случае) ≈ **16 \* q \* n \* m**

fr = 2 + q \* (2 + 4 + (m/2) \* (4 + 6 + 3 + 6) ) = (19 / 2) \* q \* m + 6 \* q + 2

fc = 2 + n \* (2 + 4 + (m/2) \* (4 + 6 + 3 + 6) ) = (19 / 2) \* n \* m + 6 \* n + 2

fb = 2 + q \* (2 + 2 + n \* (2 + 7 + 4 + (m/2) \* (4 + 12 + 6 + 10) ) ) = 2 + q \* (4 + n \* (13 + 16 \* m ) = 16 \* q \* n \* m + 13 \* q \* n + 4 \* q + 2

fo = 3 + [0 -(лучший случай); 2 + q \* (2 + 2 + n \* (2 + 8 + 4 + 2) ) -(худший случай)]= 3 + [0; 16 \* n \* q + 4 \* q + 2]

Вычисление трудоемкости оптимизированного алгоритма Винограда:

fa = 2 + 3 + fr + fc + fb = (17/2) \* q \* n \* m + 17 \* q \* n + (17/2) \* n \*m + (17/2) \* q \* m + 12 \* q + 11 (в худшем случае) ≈ (17/2) \* q \* n \* m ≈ **8.5 \* q \* n \* m**

( в лучшем случае: 8.5 \* q \* n \* m + 10 \* q \* n + 8.5 \* n \*m + 8.5 \* q \* m + 12 \* q + 11)

fr = 2 + q \* (2 + 2 + (m/2) \* (2 + 7 + 2) ) = (11/2) \* q \* m + 4 \* q + 2

fc = 2 + n \* (2 + 2 + (m /2) \* (2 + 7 + 2) ) = (11/2) \* n \* m + 4 \* q + 2

fb = 2 + q \* (2 + 2 + n \* (2 + 4 + 2 + (m/2) \* (2 + 8 + 5 + 2) + 1 + 1 + 7) ) = (17/2) \* q \* n \* m + 10 \* q \* n + 4 \* q + 2 + [0, 7 \* q \* n]

Как видно из получившихся расчетов, оптимизированный алгоритм Винограда почти в 2 раза менее трудоемкий, чем классический алгоритм Винограда.

### Технологическая часть

#### Требования к ПО

Программа на вход получает два числа: размерность первой матрицы, далее вводится сама матрица, затем аналогичным образом вводится вторая матрицы. При решении задачи после ввода матриц проверяется возможность умножения введенных матриц (число столбцов в первой матрице равно числу строк во второй). Результат работы программы: результирующая матрица, вычисленная по 3 алгоритмам. Программа должна вывести 3 матрицы - результаты работы трех алгоритмов.

#### Средства реализации

В данной работе использовался язык программирования Python. Для замера времени использовалась дополнительный модуль time.

#### Листинги кода

На вход каждой функции подавались 2 матрицы: mtr1 размером n1 на eq и mtr2 размером eq на m2.

В листинге 1 представлена реализация стандартного алгоритма умножения матриц.

Листинг 1:

1. def standart\_mult(mtr1, n1, mtr2, m2, eq):
2. res = [[0] \* m2 for i in range(n1)]
3. for i in range(n1):
4. for j in range(m2):
5. for k in range(eq):
6. res[i][j] = res[i][j] + mtr1[i][k] \* mtr2[k][j]
7. return res

В листинге 2 представлена реализация алгоритма Винограда умножения матриц.

Листинг 2:

1. def vin\_mult(mtr1, n1, mtr2, m2, eq):
2. res = [[0] \* m2 for i in range(n1)]
3. mul\_row = [0] \* n1
4. mul\_column = [0] \* m2
5. for i in range(n1):
6. for k in range(eq // 2):
7. mul\_row[i] = mul\_row[i] + mtr1[i][2 \* k] \* mtr1[i][2 \* k + 1]
8. for i in range(m2):
9. for k in range(eq // 2):
10. mul\_column[i] = mul\_column[i] + mtr2[2 \* k][i] \* mtr2[2 \* k + 1][i]
11. for i in range(n1):
12. for j in range(m2):
13. res[i][j] = - mul\_row[i] - mul\_column[j]
14. for k in range(eq // 2):
15. res[i][j] = res[i][j] + (mtr1[i][2 \* k] + mtr2[2 \* k + 1][j]) \* (mtr1[i][2 \* k + 1] + mtr2[2 \* k][j])
16. if eq % 2:
17. for i in range(n1):
18. for j in range(m2):
19. res[i][j] = res[i][j] + mtr1[i][eq - 1] \* mtr2[eq - 1][j]
20. return res

В листинге 3 представлена реализация оптимизированного алгоритма Винограда умножения матриц.

Листинг 3:

1. def opt\_vin\_mult(mtr1, n1, mtr2, m2, eq):
2. res = [[0] \* m2 for i in range(n1)]
3. mul\_row = [0] \* n1
4. mul\_column = [0] \* m2
5. d = eq - 1
6. eq2 = eq % 2
7. for i in range(n1):
8. for k in range(0, d, 2):
9. mul\_row[i] -= mtr1[i][k] \* mtr1[i][k + 1]
10. for i in range(m2):
11. for k in range(0, d, 2):
12. mul\_column[i] -= mtr2[k][i] \* mtr2[k + 1][i]
13. for i in range(n1):
14. for j in range(m2):
15. buf = mul\_row[i] + mul\_column[j]
16. for k in range(0, d, 2):
17. buf += (mtr1[i][k] + mtr2[k + 1][j]) \* (mtr1[i][k + 1] + mtr2[k][j])
18. if eq2:
19. buf += mtr1[i][d] \* mtr2[d][j]
20. res[i][j] = buf
21. return res

### Экспериментальная часть

#### Примеры работы программы

Далее на рисунках 4.1 - 4.4 будут представлены примеры работы программы на различных входных данных.

|  |
| --- |
|  |
| *рис. 4.1 - пример работы программы для матриц 2x3 и 3x2* |

|  |
| --- |
|  |
| *рис. 4.2 - пример работы программы для матриц 3x3 и 3x3* |

|  |
| --- |
|  |
| *рис. 4.3 - пример работы программы для матриц 2x2 и 2x2* |

|  |
| --- |
|  |
| *рис. 4.4 - пример работы программы для матриц 2x2 и 3x2* |

#### Результаты функционального тестирования

Ниже в таблице 1 представлены результаты функционального тестирования программы.

Таблица 1

*Тестовые случаи (классы эквивалентности)*

|  |  |  |  |
| --- | --- | --- | --- |
| mtr1 | mtr2 | Ожидаемый результат | Полученный результат |
| 1 | 2 | 2 | 2 |
| 1 2 | 2  1 | 4 | 4 |
| 1 -2  -3 4 | 5 6  7 -8 | -9 22  13 -50 | -9 22  13 -50 |
| 4 -1 3  4 -2 -6  2 0 3 | 5 3 -7  -1 6 -3  2 -4 1 | 27 -6 -22  10 24 -28  16 -6 -11 | 27 -6 -22  10 24 -28  16 -6 -11 |
| 2 -3 1  5 4 -2 | -7 5  2 -1  4 3 | -16 16  -35 15 | -16 16  -35 15 |
| 1 2  3 4 | 2 7 1  6 9 -7  2 3 4 | Умножение матриц невозможно | Умножение матриц невозможно |

В таблице выше в колонке “Полученный результат” представлен результат работы программы, при этом этот результат совпадает во всех трех алгоритмов.

Во время тестирования программа верно вычислила результат при вводе матриц с некорректными (для операции умножения) размерами, а также при вводе квадратных матриц с четным и нечетным значением размера.

#### Постановка эксперимента по замеру времени

При сравнении быстродействия алгоритмов были использованы матрицы размерами в диапазоне от 100x100 до 1000x1000 с шагом 100 (и по строкам и по столбцам) и в диапазоне от 101x101 до 1001x1001 также с шагом 100. Результат одного эксперимента рассчитывался как средний из результатов проведенных испытаний с одинаковыми входными данными. Количество повторов каждого эксперимента = 10. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

#### Сравнительный анализ на материале экспериментальных данных

Ниже на рис. 4.5 - 4.6 приведены графики зависимости временных затрат работы алгоритмов (в секундах) от размеров матриц.

|  |
| --- |
|  |
| *рис. 4.5 - график зависимости времени работы программы от четных входных размеров матриц* |

На графике видно, что оптимизированная реализация алгоритма Винограда примерно на 25% быстрее реализации стандартного алгоритма, в то же время стандартная реализация быстрее классической реализации алгоритма Винограда примерно на 15%.

|  |
| --- |
|  |
| *рис. 4.6 - график зависимости времени работы программы от нечетных входных размеров матриц* |

На графике видно, что оптимизированная реализация алгоритма Винограда примерно на 23% быстрее реализации стандартного алгоритма, в то же время стандартная реализация быстрее классической реализации алгоритма Винограда примерно на 15%.

В результате проведенного эксперимента был получен следующий вывод: оптимизированный алгоритм Винограда эффективнее двух других анализируемых алгоритмов и при четных, и при нечетных размерах входных матриц. При этом разница во времени работы программы при четных и нечетных размерах пренебрежимо мала. Стандартный алгоритм работает за одно и то же время вне зависимости от размеров входных матриц.

### Заключение

Цель данной лабораторной работы была достигнута. В ходе работы были изучены алгоритмы умножения матриц: стандартный и Винограда (классический и оптимизированный). Данные алгоритмы были реализованы и была вычислена трудоемкость каждого алгоритма. Был проведен сравнительный анализ перечисленных выше алгоритмов, а также были экспериментально получены зависимости времени выполнения алгоритмов от размеров входных матриц.