**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Анализ алгоритмов**

**Лабораторная работа №2**

**Отчёт на тему:**

# «Умножение матриц»

Выполнила:

Янова Даниэлла

ИУ7-53

**Москва, 2018**

**Введение**

Алгоритм Копперсмита—Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом (англ.). В исходной версии асимптотическая сложность алгоритма составляла O(n2,3755), где n — размер стороны матрицы. Алгоритм Копперсмита—Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц[1].

На практике алгоритм Копперсмита—Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров[2].

Поэтому на практике обычно пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости.

Алгоритм Штрассена предназначен для быстрого умножения матриц. Он был разработан Фолькером Штрассеном в 1969 году и является обобщением метода умножения Карацубы на матрицы.

В отличие от традиционного алгоритма умножения матриц, алгоритм Штрассена умножает матрицы за время Θ(n log2 7 ) = O(n 2.81)Θ(n log2 7 ) = O(n 2.81).

Это даёт выигрыш на больших плотных матрицах начиная, примерно, от 64 на 64.

Несмотря на то, что алгоритм Штрассена является асимптотически не самым быстрым из существующих алгоритмов быстрого умножения матриц, он проще программируется и эффективнее при умножении матриц относительно малого размера, поэтому именно он чаще используется на практике.

**Задачи работы**

Реализовать алгоритмы умножения матриц, описанные ниже.

1. Классический алгоритм умножения.
2. Алгоритм Копперсмита—Винограда.
3. Улучшенный Алгоритм Копперсмита—Винограда

**Аналитическая часть**

В данном разделе приведены алгоритмы, а также составлена модель для вычисления трудоемкости.

**1.1 Умножение матриц. Стандартный алгоритм**

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является прямоугольной (в частных случаях – квадратной) таблицей, совокупностью строк и столбцов, на пересечении которых находятся элементы матрицы. Количество строк и столбцов является размерностью матриц. Для матриц определена операция умножения.

Пусть даны две матрицы – матрица А размером m на n и матрица B размером n на l.

A =, B = ,

Тогда матрица C размерностью g ×n

C = ,

в которой (i = 1, 2, . . . l; j = 1, 2, . . . n) называется их ”произведением”. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы ”согласованы”. В частности, умножение всегда выполнимо, если оба сомножителя — [[Квадратная матрица|квадратные матрицы]] одного и того же порядка.

Таким образом, из существования произведения AB вовсе не следует существование произведения BA

**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. Рассмотрим два вектора

V = (v1, v2, v3, v4) и W = (w1, w2, w3, w4).

Их скалярное произведение равно:

Это равенство можно переписать в виде:

Кажется, что второе выражение задает больше работы, чем первое: 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 4 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.

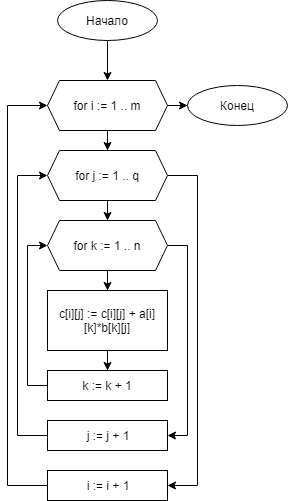
**Улучшения алгоритма**

1. В 2010 Эндрю Стотерс усовершенствовал алгоритм до O(n 2.374).
2. В 2011 году Вирджиния Вильямс усовершенствовала алгоритм ещё раз — O(n 2.3728642).
3. В 2014 году Франсуа Ле Галль упростил метод Уильямс и получил новую улучшенную оценку O(n 2.3728639).

2. **Конструкторская часть**

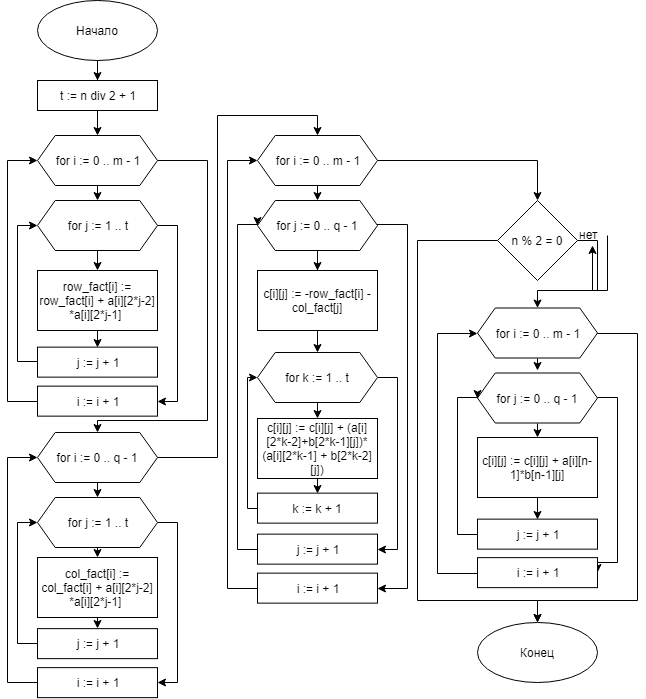
В данном разделе представлены схемы алгоритмов.

* 1. **Стандартный алгоритм**

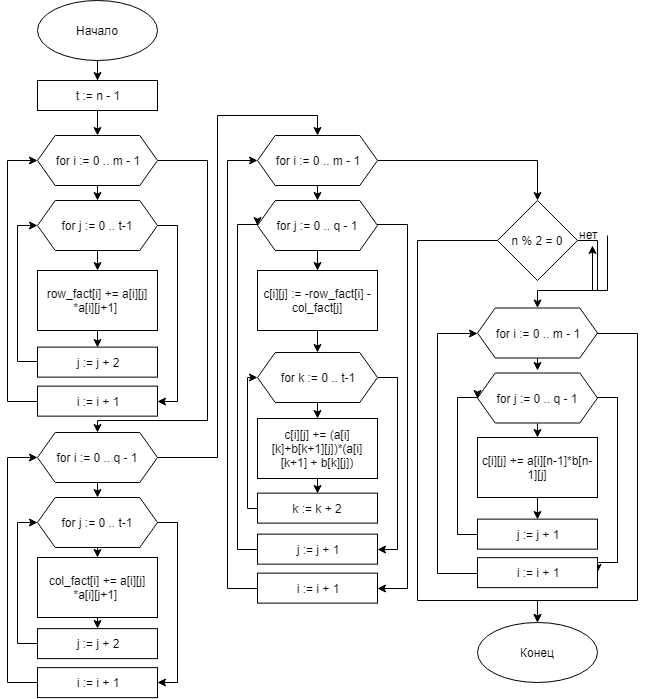
****

**Рисунок 1.1 – Схема стандартного алгоритма**

* 1. **Алгоритм Винограда**

**Рисунок 1.2 – Схема алгоритма Винограда**

* 1. **Оптимизированный алгоритм Винограда**

**Рисунок 1.3 – Схема оптимизированного алгоритма Винограда**

**3. Технологическая часть**

В данном разделе представлена реализация алгоритмов, указан язык прораммирования, а также необходимые модули.

**3.1 Средства реализации**

Для выполнения данной лабораторной работы использовался язык Python 3.7.1 в среде Pycharm. Замены времени проводились с использованием функции process\_time\_ns, входящей в библиотеку time Python версии 3.7.

**3.2 Листинг кода**

**def** standart\_matrix\_mult(a, b):

m = len(a)

n = len(a[0])

q = len(b[0])

c = [[0 **for** x **in** range(q)] **for** y **in** range(m)]

**for** i **in** range(m):

**for** j **in** range(q):

**for** k **in** range(n):

c[i][j] = c[i][j] + a[i][k]\*b[k][j]

**return** c

Листинг 1.1 – Стандартный алгоритм умножения матриц

**def** **classic\_vinograd\_mult**(a, b):

m = len(a)

n = len(a[**0**])

q = len(b[**0**])

c = [[**0** **for** x **in** range(q)] **for** y **in** range(m)]

row\_fact = [**0** **for** x **in** range(m)]

column\_fact = [**0** **for** x **in** range(q)]

**for** i **in** range(m):

row\_fact[i] = **0**

**for** j **in** range(**1**, n//**2**+**1**):

row\_fact[i] = row\_fact[i] + a[i][**2**\*j-**2**] \* a[i][**2**\*j-**1**]

**for** i **in** range(q):

column\_fact[i] = **0**

**for** j **in** range(**1**, n//**2**+**1**):

column\_fact[i] = column\_fact[i] + b[**2**\*j-**2**][i] \* b[**2**\*j-**1**][i]

**for** i **in** range(m):

**for** j **in** range(q):

c[i][j] = - row\_fact[i] - column\_fact[j]

**for** k **in** range(**1**, n//**2**+**1**):

c[i][j] = c[i][j] + (a[i][**2**\*k-**2**]+b[**2**\*k-**1**][j])\*(a[i][**2**\*k-**1**] + b[**2**\*k-**2**][j])

**if** (n % **2** == **1**):

**for** i **in** range(m):

**for** j **in** range(q):

c[i][j] = c[i][j] + a[i][n-**1**]\*b[n-**1**][j]

**return c**

Листинг 1.2 – Стандартный алгоритм Винограда

**def** **optimized\_vinograd\_mult**(a, b):

m = len(a)

n = len(a[**0**])

q = len(b[**0**])

c = [[**0** **for** x **in** range(q)] **for** y **in** range(m)]

t = n-**1**

row\_fact = [**0** **for** x **in** range(m)]

column\_fact = [**0** **for** x **in** range(q)]

**for** i **in** range(m):

**for** j **in** range(**0**,t,**2**):

row\_fact[i] += a[i][j] \* a[i][j+**1**]

**for** i **in** range(q):

**for** j **in** range(**0**,t,**2**):

column\_fact[i] += b[j][i] \* b[j+**1**][i]

**for** i **in** range(m):

**for** j **in** range(q):

c[i][j] = -row\_fact[i] - column\_fact[j]

**for** k **in** range(**0**,t,**2**):

c[i][j] += (a[i][k]+b[k+**1**][j])\*(a[i][k+**1**] + b[k][j])

**if** (n % **2**):

**for** i **in** range(m):

**for** j **in** range(q):

c[i][j] += a[i][n-**1**]\*b[n-**1**][j]

**return** c

Листинг 1.3 – Оптимизированный алгоритм Винограда

1. **Экспериментальная часть**

В данном разделе будут приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

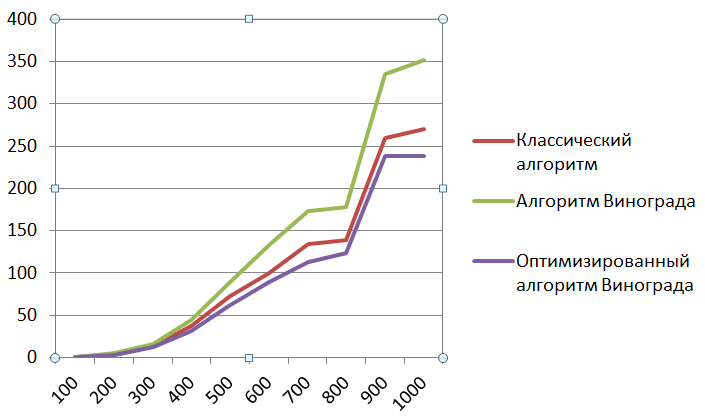
**4.1 Примеры работы**

**4.2 Тестирование алгоритмов**

Были проведены исследования зависимости времени работы трех алгоритмов от размеров перемножаемых матриц. Замеры времени проводились отдельно для матриц четной размерности и матриц нечетной размерности. Для эксперимента использовались матрицы размера от 100 до 1000 с шагом 100 для четной размерности и размера от 101 до 1001 с шагом 100 для нечетной размерности. Временные замеры проводятся путём многократного проведения эксперимента и деления результирующего времени на количество итераций эксперимента. Замены времени проводились с использованием функции process\_time\_ns, входящей в библиотеку time Python версии 3.7.

**Матрицы четного размера.**

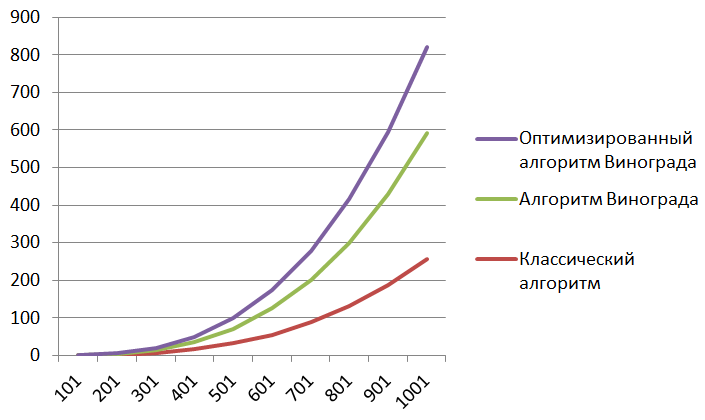
|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Классический алгоритм | Алгоритм Винограда | Оптимизированный  алгоритм Винограда |
| 100 | 0.4390625 | 0.46875 | 0.378125 |
| 200 | 4.0703125 | 4.725000 | 3.2671875 |
| 300 | 13.371875 | 15.50625 | 11.7203125 |
| 400 | 36.759375 | 43.6765625 | 31.634375 |
| 500 | 72.3734375 | 88.6609375 | 62.4203125 |
| 600 | 100.362500 | 133.3484375 | 89.5734375 |
| 700 | 134.246875 | 173.50625 | 113.3390625 |
| 800 | 138.7140625 | 177.296875 | 123.471875 |
| 900 | 259.2859375 | 335.0203125 | 238.6921875 |
| 1000 | 269.809375 | 351.3484375 | 237.840625 |



**Рисунок 4.1 – Замеры времени для матриц четного размера**

**Матрицы нечетного размера.**

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Классический алгоритм | Алгоритм Винограда | Оптимизированный  алгоритм Винограда |
| 101 | 0.2515625 | 0.2828125 | 0.2140625 |
| 201 | 1.9796875 | 2.2406250 | 1.6750000 |
| 301 | 6.7453125 | 7.8203125 | 5.7750000 |
| 401 | 16.178125 | 19.381250 | 14.065625 |
| 501 | 31.8578125 | 39.0078125 | 27.796875 |
| 601 | 55.228125 | 70.365625 | 48.325000 |
| 701 | 87.5296875 | 112.040625 | 77.0796875 |
| 801 | 131.221875 | 168.9796875 | 115.515625 |
| 901 | 186.1171875 | 242.3296875 | 164.971875 |
| 1001 | 256.9203125 | 336.1453125 | 228.862500 |



**Рисунок 4.2 – Замеры времени для матриц нечетного размера**

**4.3 Вывод**

Алгоритм Винограда начинает выигрывать в быстродействии у других алгоритмов только в случае, когда матрицы имеют размерность, не позволяющую хранить их целиком в памяти компьютера. Стандартная реализация алгоритма Винограда работает дольше, чем стандартный алгоритм умножения матриц, в то время как оптимизированная версия алгоритма Винограда работает быстрее их обоих.

**Заключение**

В ходе работы были изучены и реализованы алгоритмы классического умножения матриц , алгоритма Виноградова и алгоритма Виноградова с оптимизациями. Также был проведен сравнительный анализ перечисленных алгоритмов.. Экспериментально подтверждено различие во временнoй эффективности. Классический алгоритм работает медленнее, чем алгоритм Винограда или его оптимизированная версия, поскольку если обычный алгоритм работает за 13\*n3 (с учетом, что на вход пришли две квадратных матрицы размера n), то алгоритм Винограда работает за 13n3, а его улучшенная версия 9\*n3. При этом с точки зрения объема затрачиваемой памяти, он более эффективен, поскольку не выделяет память под дополнительные массивы, как это делает алгоритм Винограда. Если же говорить об обычном и оптимизированным, то оптимизированный работает быстрее за счет занесения вычисления B/2 в отдельную переменную и измененный пересчет вспомогательных строк r и c.

**Список литературы**:

[1] Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Chris Umans. Grouptheoretic Algorithms for Matrix Multiplication. arXiv:math.GR/0511460. Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379—388..

[2] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.

[3] И.В. Ломовской. Курс лекций по языку программирования C, 2017