|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 №** | 02 |

**Название:**

***Умножение матриц***

**Дисциплина:  *Анализ алгоритмов***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-56Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Волокова Л. Л.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2020*

**Оглавление**

[Введение 3](#_Toc56263191)

[I. Аналитическая часть 4](#_Toc56263192)

[1. Задачи 4](#_Toc56263193)

[2. Описание алгоритмов 4](#_Toc56263194)

[*i.* *Классический алгоритм умножения* 4](#_Toc56263195)

[*ii.* *Алгоритм Винограда* 5](#_Toc56263196)

[*iii.* *Оптимизированный алгоритм Винограда* 6](#_Toc56263197)

[3. Модель вычислений 6](#_Toc56263198)

[II. Конструкторская часть 7](#_Toc56263199)

[1. Схемы алгоритмов 7](#_Toc56263200)

[2. Оценка трудоёмкости 10](#_Toc56263201)

[*i.* *Классический алгоритм:* 10](#_Toc56263202)

[*ii.* *Алгоритм Винограда:* 10](#_Toc56263203)

[*iii.* *Оптимизированный алгоритм Винограда* 10](#_Toc56263204)

[III. Технологическая часть 11](#_Toc56263205)

[1. Требования к программному обеспечению: 11](#_Toc56263206)

[2. Средства реализации 11](#_Toc56263207)

[3. Реализации алгоритмов 11](#_Toc56263208)

[*i.* *Классический алгоритм* 11](#_Toc56263209)

[*ii.* *Алгоритм Винограда* 12](#_Toc56263210)

[*iii.* *Оптимизированный алгоритм Винограда* 12](#_Toc56263211)

[4. Тесты 13](#_Toc56263212)

[IV. Экспериментальная часть 14](#_Toc56263213)

[1. Примеры работы 14](#_Toc56263214)

[2. Сравнение работы алгоритмов при чётных размерах матрицы 14](#_Toc56263215)

[3. Сравнение работы алгоритмов при нечётных размерах матрицы 15](#_Toc56263216)

[Заключение 17](#_Toc56263217)

# Введение

В огромном количестве областей научной и технической сферы деятельности человека при различных математических расчетах используют такую операцию как умножение матриц. Это довольно трудоемкий процесс даже при небольших размерах матриц, так как требуется большое количество операций умножения и сложения различных чисел. По этой причине человек озадачен проблемой оптимизации умножения матриц и ускорения процесса вычисления.

Таким образом, эффективное умножение матриц по времени и затратам ресурсов является актуальной проблемой для науки и техники.

1. **Аналитическая часть**
   1. **Задачи**

**Цель лабораторной работы** - изучение трех алгоритмов умножения матриц: классического, алгоритма Винограда и его оптимизации.

Для того чтобы добиться этой цели, были поставлены следующие задачи:

* изучить и реализовать классический алгоритм умножения матриц и алгоритм Винограда
* оптимизировать работу алгоритма Винограда
* выполнить сравнительный анализ трудоёмкостей алгоритмов
* сравнить эффективность алгоритмов по времени
  1. **Описание алгоритмов**
     1. *Классический алгоритм умножения*

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является таблицей, на пересечении строк и столбцов находятся элементы матрицы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две прямоугольные матрицы A и B размерности N \* M, M \* Q соответственно:

Тогда произведением матриц A и B называется матрица C размерностью M\*Q:

* + 1. *Алгоритм Винограда*

Рассматривая результат умножения двух матриц очевидно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора:

*где U = Ai - i-ая строка матрицы A,*

*V = Bj - j-ый столбец матрицы B*

Их скалярное произведение равно:

Это равенство можно переписать в виде:

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: 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, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного.

* + 1. *Оптимизированный алгоритм Винограда*

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

* предварительные вычисления повторяющихся одинаковых действий
* уменьшения количества повторных проверок;
* замена цикл **for (int i = 0; i < N / 2; i++)** на **for (int i = 0; i< N – 1; i + =2)**
  1. Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

* +, -, /, \%, ==, !=, <, >, <=, >=, [], ++, \-- -- имеют трудоемкость 1
* трудоемкость оператора выбора

if (условие) then

A

Else

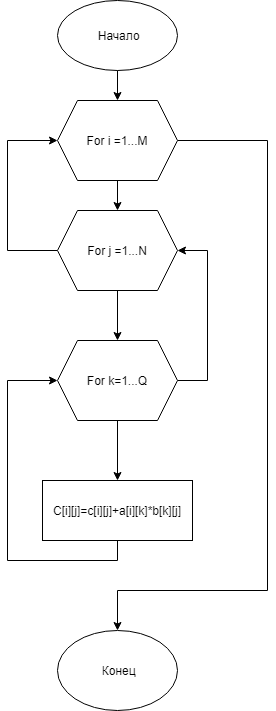
B

рассчитывается, как

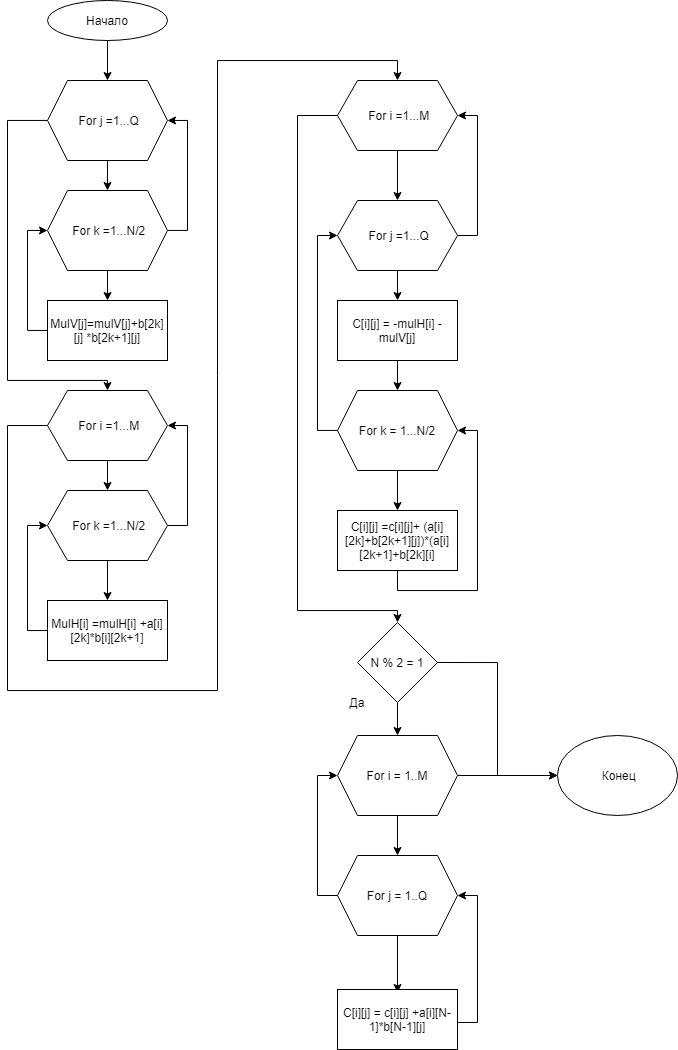
* трудоемкость цикла рассчитывается, как
* трудоемкость вызова метода равна 0
* трудоемкость вызова функции равна 1

1. **Конструкторская часть**
   1. **Схемы алгоритмов**

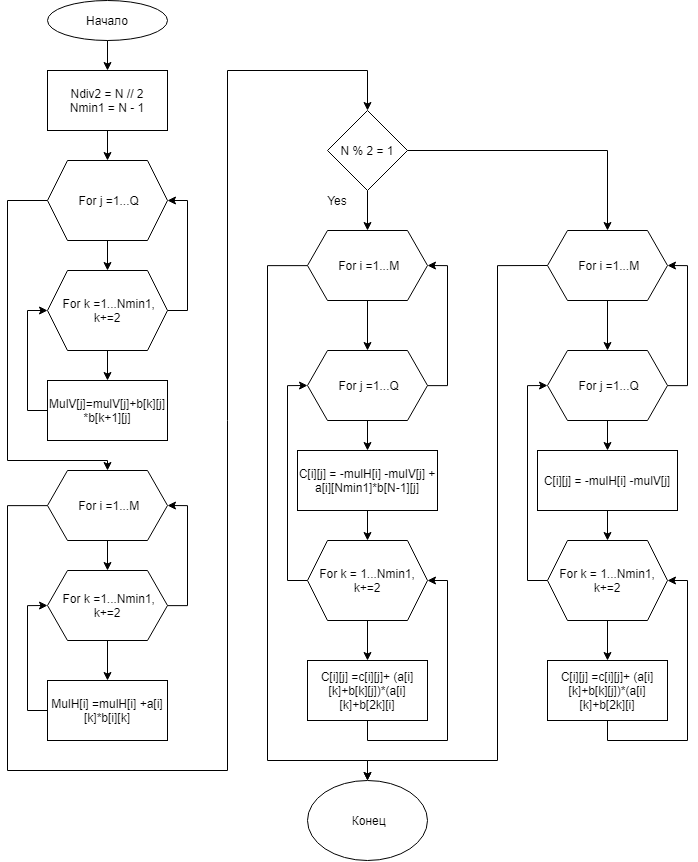
**Классический алгоритм**

****

**Алгоритм Винограда**

****

**Оптимизированный алгоритм Винограда**

****

* 1. **Оценка трудоёмкости**

Пусть даны две матрицы A и B размерностью MxN и размерностью NxQ соответственно. Рассмотрим трудоемкость трёх алгоритмов умножения матриц.

* + 1. *Классический алгоритм:*
    2. *Алгоритм Винограда:*

Общая трудоемкость алгоритма:

* + 1. *Оптимизированный алгоритм Винограда*

+

1. **Технологическая часть**
   1. **Требования к программному обеспечению:**

На вход подаются размеры двух матриц. Матрицы генерируются случайным образом и выводятся на экран. На выход программа выдаёт три матрицы, которые являются результатами работы трёх различных алгоритмов умножения.

* 1. **Средства реализации**

Для реализации программы был использован язык Python. Для замера процессорного времени была использована функция time() из библиотеки time.

* 1. **Реализации алгоритмов**
     1. *Классический алгоритм*

1. **def** matrix\_mult\_stand(MatA, MatB):
2. **if** MatA.col != MatB.row:
3. **return** Mat()
5. M = MatA.row
6. N = MatA.col
7. Q = MatB.col
8. MatC = matrix\_create(M, Q)
10. **for** i **in** range(M):
11. **for** j **in** range(Q):
12. **for** k **in** range(N):
13. MatC.mat[i][j] = MatC.mat[i][j] + MatA.mat[i][k] \* MatB.mat[k][j]
15. **return** MatC
    * 1. *Алгоритм Винограда*
16. **def** matrix\_mult\_winograd(A, B):
17. **if** A.col != B.row:
18. **return** Mat()
20. M = A.row
21. N = A.col
22. Q = B.col
23. # I
24. mulV = [0 **for** i **in** range(Q)]
25. **for** j **in** range(Q):
26. **for** k **in** range(N // 2):
27. mulV[j] = mulV[j] + B.mat[2 \* k][j] \* B.mat[2 \* k + 1][j]
29. #II
30. mulH = [0 **for** i **in** range(M)]
31. **for** i **in** range(M):
32. **for** k **in** range(N // 2):
33. mulH[i] = mulH[i] + A.mat[i][2 \* k] \* A.mat[i][2 \* k + 1]
35. #III
36. C = matrix\_create(M, Q)
37. **for** i **in** range(M):
38. **for** j **in** range(Q):
39. C.mat[i][j] = -mulH[i] - mulV[j]
40. **for** k **in** range(N // 2):
41. C.mat[i][j] = C.mat[i][j] + (A.mat[i][2 \* k] + B.mat[2 \* k + 1][j]) \* (A.mat[i][2 \* k + 1] + B.mat[2 \* k][j])
43. #IV
44. **if** (N % 2 == 1):
45. **for** i **in** range(M):
46. **for** j **in** range(Q):
47. C.mat[i][j] = C.mat[i][j] + A.mat[i][N - 1] \* B.mat[N - 1][j]
49. **return** C
    * 1. *Оптимизированный алгоритм Винограда*
50. **def** matrix\_mult\_5(A, B):
52. **if** A.col != B.row:
53. **return** Mat()
54. M = A.row
55. N = A.col
56. Q = B.col
57. Ndiv2 = N // 2
58. Nmin1 = N - 1
60. # I
61. mulV = [0 **for** i **in** range(Q)]
62. **for** j **in** range(Q):
63. **for** k **in** range(0, Nmin1, 2):
64. mulV[j] += B.mat[k][j] \* B.mat[k + 1][j]
66. #II
67. mulH = [0 **for** i **in** range(M)]
68. **for** i **in** range(M):
69. **for** k **in** range(0, Nmin1, 2):
70. mulH[i] += A.mat[i][k] \* A.mat[i][k + 1]
72. #III
73. C = matrix\_create(M, Q)
75. **if** (N % 2 == 1):
76. **for** i **in** range(M):
77. **for** j **in** range(Q):
78. C.mat[i][j] = -mulH[i] - mulV[j] + A.mat[i][Nmin1] \* B.mat[Nmin1][j]
79. **for** k **in** range(0, Nmin1, 2):
80. C.mat[i][j] += (A.mat[i][k] + B.mat[k + 1][j]) \* (A.mat[i][k + 1] + B.mat[k][j])

83. **else**:
84. **for** i **in** range(M):
85. **for** j **in** range(Q):
86. C.mat[i][j] = -mulH[i] - mulV[j]
87. **for** k **in** range(0, Nmin1, 2):
88. C.mat[i][j] += (A.mat[i][k] + B.mat[k + 1][j]) \* (A.mat[i][k + 1] + B.mat[k][j])

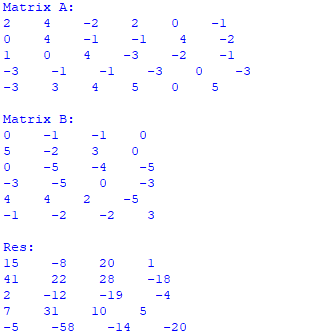
91. **return** C
    1. **Тесты**

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице:

|  |  |  |
| --- | --- | --- |
| A | B | C = AxB |
| [3] | [-5] | [-15] |
|  |  |  |
|  |  |  |
|  |  |  |

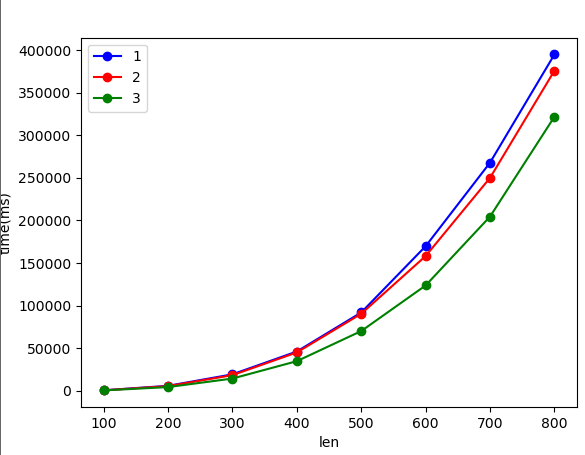
В результате проверки реализации всех алгоритмов умножения прошли все поставленные функциональные тесты.

1. **Экспериментальная часть**
   1. **Примеры работы**



* 1. **Сравнение работы алгоритмов при чётных размерах матрицы**

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 100 до 800 с шагом 100. Результаты измерений показаны в таблице и на рисунке.

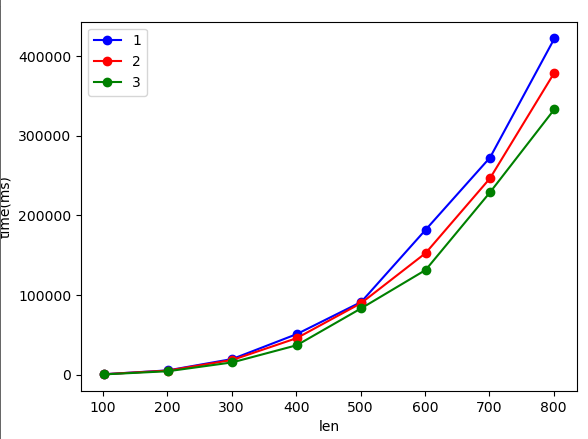


|  |  |  |  |
| --- | --- | --- | --- |
|  | **Классический алгоритм** | **Алгоритм Винограда** | **Оптимизированный алгоритм Винограда** |
| **100** | 706 | 670 | 579 |
| **200** | 5942 | 5596 | 4472 |
| **300** | 19420 | 18470 | 14479 |
| **400** | 46200 | 45099 | 34869 |
| **500** | 92010 | 90497 | 70195 |
| **600** | 169787 | 158238 | 123730 |
| **700** | 267907 | 250059 | 204595 |
| **800** | 394933 | 375859 | 321825 |

Из результатов экспериментов можно сделать вывод о том, что алгоритм Винограда выигрывает классический алгоритм умножения. Оптимизированный алгоритм работает быстрее обычного алгоритма Винограда.

* 1. **Сравнение работы алгоритмов при нечётных размерах матрицы**

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 101 до 801 с шагом 100. Результаты измерений показаны в таблице



|  |  |  |  |
| --- | --- | --- | --- |
|  | **Классический алгоритм** | **Алгоритм Винограда** | **Оптимизированный алгоритм Винограда** |
| **100** | 710 | 699 | 582 |
| **200** | 5996 | 5612 | 4520 |
| **300** | 19882 | 18669 | 15639 |
| **400** | 50132 | 46080 | 38145 |
| **500** | 91433 | 90059 | 87468 |
| **600** | 182053 | 160794 | 131593 |
| **700** | 272717 | 253713 | 225085 |
| **800** | 412263 | 379459 | 333430 |

Для случая с нечётными размерами матриц можно сделать те же выводы, что и для случая с чётными. При этом можно заметить, что классический алгоритм в среднем работает за то же время, что и при чётных размерах, в то время как алгоритм Винограда и его оптимизация работают дольше за счёт дополнительных операций при нечётном случае. Однако по-прежнему классический алгоритм проигрывает по времени на те же величины.

# Заключение

В ходе лабораторной работе были изучены и реализованы три алгоритма умножения матриц: классический алгоритм, алгоритм Винограда и его оптимизированный вариант. Сравнительный анализ алгоритмов показал, что алгоритмы Винограда, введя дополнительные векторы, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный и оптимизированный варианты работают быстрее классического алгоритма.