Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №2

по курсу «Анализ алгоритмов»

Умножение матриц

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Оглавление

Введение

В настоящее время у современного человека очень много различных задач. Он научился запускать ракеты в космос, строить различные машины и станки, строить здания колоссальной высоты. Но для всего этого человеку необходимо выполнять необходимые расчеты, чтобы обеспечить безопасность. В огромном количестве областей при расчетах используют такое математическое действие как умножение матриц. Это довольно трудоемкий процесс, именно поэтому человек озадачен проблемой его оптимизации.
Таким образом, умножение матриц является актуальной проблемой в настоящее время, а эффективное умножение матриц - еще более актуальная задача.

1. Аналитическая часть

1.1 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 и памяти.

1.2 Описание алгоритмов

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 $m \times n$, $n \times q$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nq} \end{bmatrix}$$

Тогда произведением матриц A и B называется матрица C размерностью $m \times q$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix}, \tag{1.1}$$

в которой:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \left(\overline{i = 1 \dots m};\ \overline{j = 1 \dots q}\right).$$

1.2.1 Алгоритм Винограда

Исходя из равенства 1.1, видно, 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 U и V:

$$U = A_i = (u_1, u_2, \dots, u_n), \tag{1.2}$$

где $U = A_i$ – i-ая строка матрицы A,
$u_k = a_{ik}, \overline{k = 1 \dots n}$ – элемент i-ой строки k-ого столбца матрицы A.

$$V = B_j = (v_1, v_2, \dots, v_n), \tag{1.3}$$

где $V = B_j$ – j-ый столбец матрицы B,
$v_k = b_{kj}, \overline{k = 1 \dots n}$ – элемент k-ой строки j-ого столбца матрицы B.

По определению их скалярное произведение равно:

$$U \cdot V = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4. \tag{1.4}$$

Равенство 1.4 можно переписать в виде:

$$U \cdot V = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1 u_2 - u_3 u_4 - v_1 v_2 - v_3 v_4. \tag{1.5}$$

В равенстве 1.4 насчитывается 4 операции умножения и 3 операции сложения, в равенстве 1.5 насчитывается 6 операций умножения и 9 операций сложения. Однако выражение $-u_1 u_2 - u_3 u_4$ используются повторно при умножении i-ой строки матрицы A на каждый из столбцов матрицы B, а выражение $-v_1 v_2 - v_3 v_4$ - при умножении j-ого столбца матрицы B на строки матрицы A. Таким образом, данные выражения можно вычислить предварительно для каждых строк и столбцов матриц для сокращения повторных вычислений. В результате повторно будут выполняться лишь 2 операции умножения и 7 операций сложения (2 операции нужны для добавления предварительно посчитанных произведений).

1.2.2 Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

- предварительные вычисления повторяющихся одинаковых действий;
- использование более быстрых операций при вычислении (такие, как смещение битов вместо умножения или деления на 2);
- уменьшения количества повторных проверок;
- использование аналогичных конструкций, уменьшающих трудоёмкость операций (к примеру, замена сложения с 1 на инкремент).

Ниже представлен список личностей, проводивших оптимизацию алгоритма:

- в 2010 Эндрю Стотерс усовершенствовал алгоритм до $O(n^{2.374})$;
- в 2011 году Вирджиния Уильямс усовершенствовала алгоритм до $O(n^{2.3728642})$;
- в 2014 году Франсуа Ле Галль упростил метод Уильямса и получил новую улучшенную оценку $O(n^{2.3728639})$.

1.2.3 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

- операции, имеющие трудоемкость 1: $<$, $>$, $=$, $<=$, $=>$, $==$, $!=$,$+$, $-$, $*$, $/$, $+=$, $-=$, $*=$, $/=$, [ ];
- оператор условного перехода имеет трудоёмкость, равную трудоёмкости операторов тела условия;
- оператор цикла for имеет трудоемкость:

$$F_{for} = F_{init} + F_{check} + N * (F_{body} + F_{inc} + F_{check}), \tag{1.6}$$

где F_{init} – трудоёмкость инициализации, F_{check} – трудоёмкость проверки условия, F_{inc} – трудоёмкость инкремента аргумента, F_{body} – трудоёмкость операций в теле цикла, N – число повторений.

2. Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов трёх алгоритмов умножения матриц.

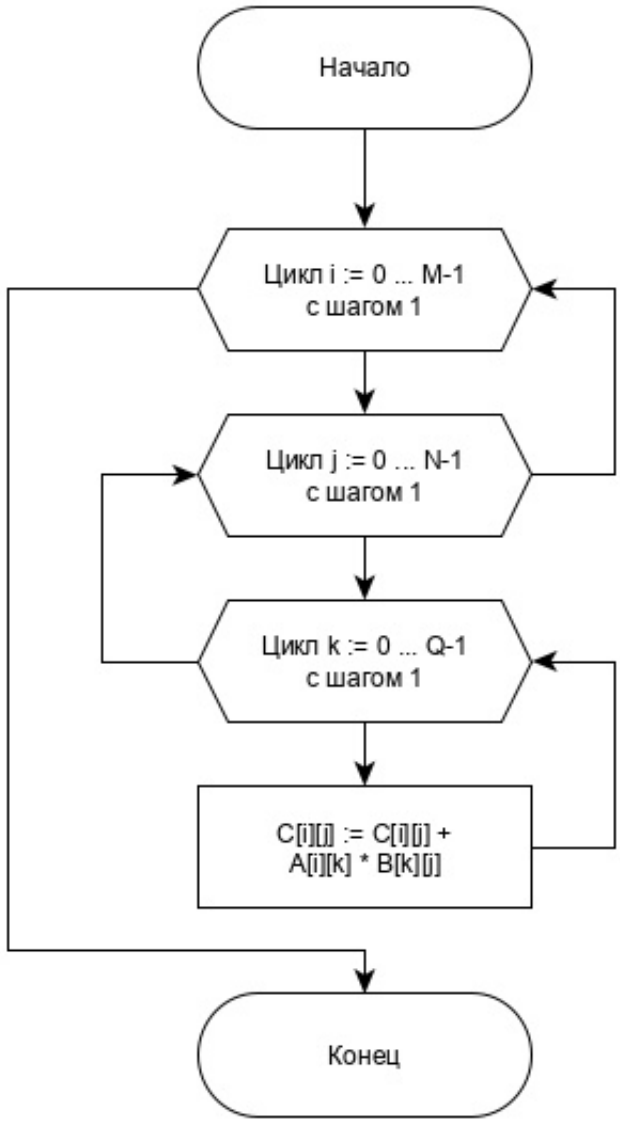

Рис. 2.1: Классический алгоритм

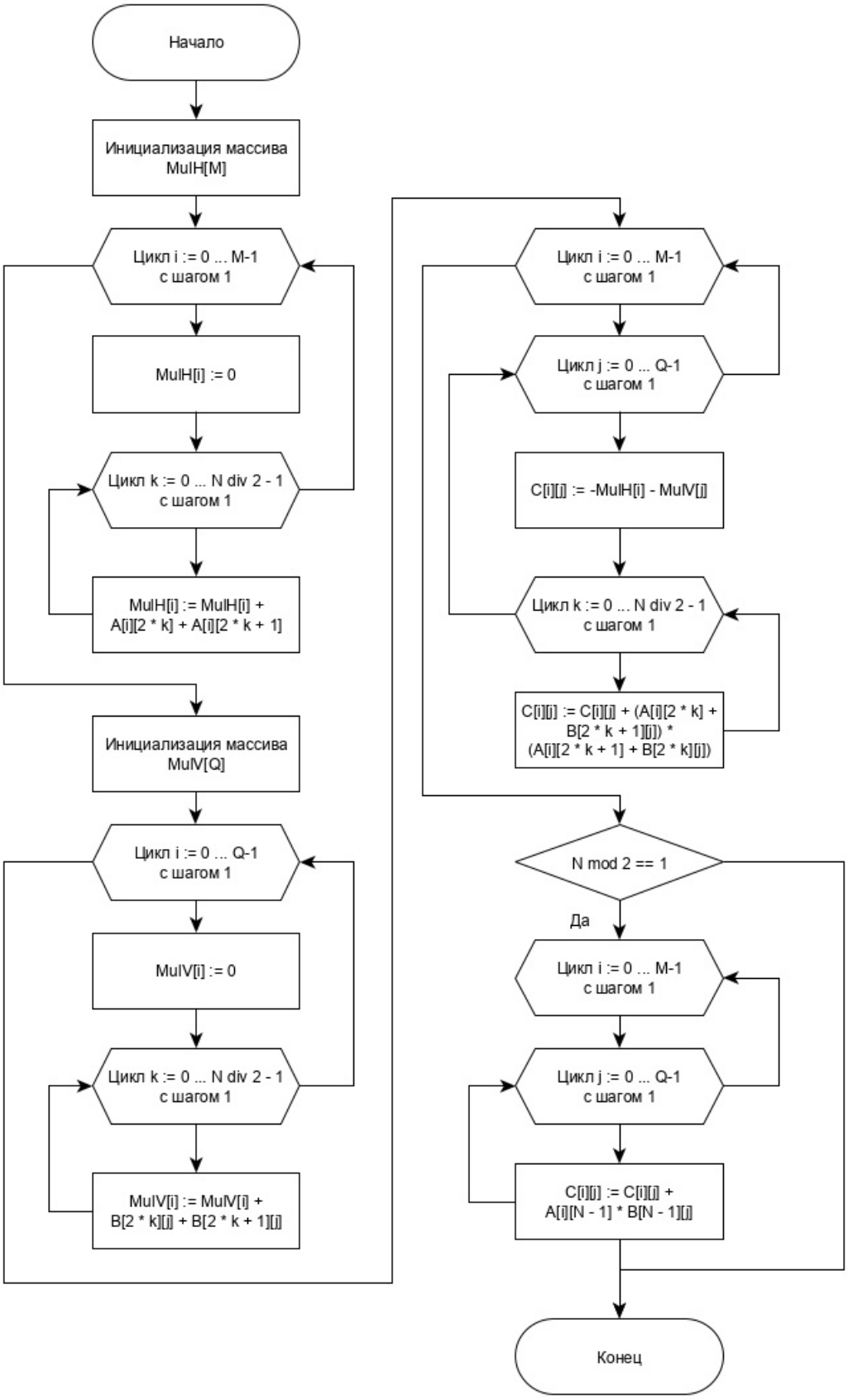

Рис. 2.2: Алгоритм Винограда

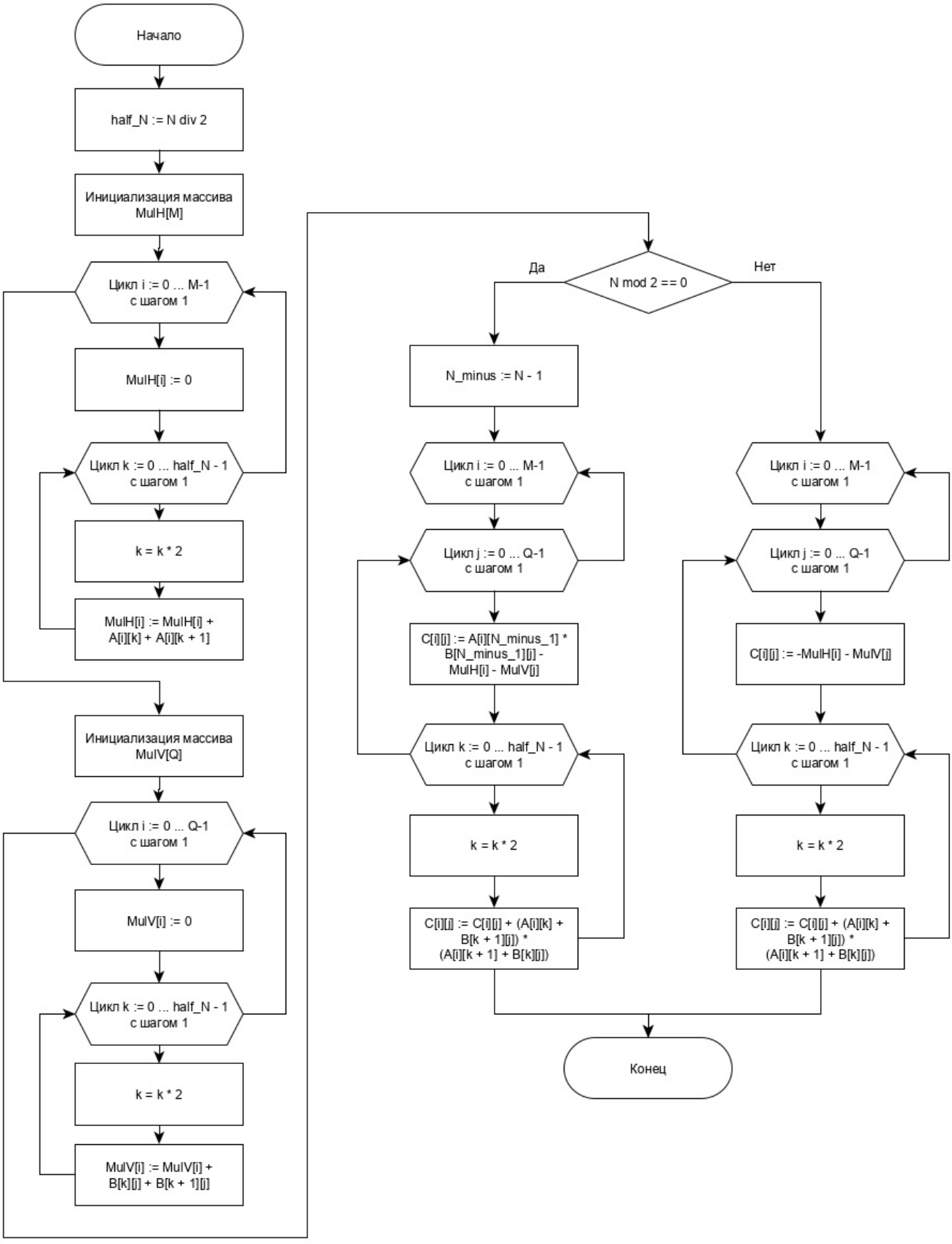

Рис. 2.3: Оптимизированный алгоритм Винограда

2.2 Оценка трудоёмкости

Пусть даны две матрицы A и B размерностью $M \times N$ и размерностью $N \times Q$ соответственно. Рассмотрим трудоемкость трёх алгоритмов умножения матриц.

2.2.1 Классический алгоритм

$$f_{classic} = 13MNQ + 4MQ + 4M + 1 \tag{2.1}$$

2.2.2 Алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH: $f_I = \frac{15}{2}MN + 4M + 2$;
- цикл создания вектора MulV: $f_{II} = \frac{15}{2}NQ + 4Q + 2$;
- основной цикл: $f_{III} = 13MNQ + 4MQ + 4M + 2$;
- условный переход при чётном N: $f_{IV} = 2$;
- условный переход и цикл при нечётном N: $f_{IV} = 17MQ + 4M + 4$.

Общая трудоемкость алгоритма:

- при чётном N:

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 4MQ + 8M + 4Q + 8 \tag{2.2}$$

- при нечётном N:

$$f_{vin} = f_I + f_{II} + f_{III} + f_{IV} = 13MNQ + \frac{15}{2}MN + \frac{15}{2}NQ + 21MQ + 12M + 4Q + 10 \tag{2.3}$$

2.2.3 Оптимизированный алгоритм Винограда

Трудоёмкости составных частей алгоритма:

- цикл создания вектора MulH: $f_I = \frac{11}{2}MN + 4M + 2$;
- цикл создания вектора MulV: $f_{II} = \frac{11}{2}NQ + 4Q + 2$;
- основной цикл при чётном N: $f_{III} = 10MNQ + 17MQ + 4M + 6$;
- основной цикл при нечётном N: $f_{III} = 10MNQ + 10NQ + 4M + 2$.

Общая трудоемкость алгоритма:

- при чётном N:

$$f_{opt} = f_I + f_{II} + f_{III} = 10MNQ + \frac{11}{2}MN + \frac{11}{2}NQ + 17MQ + 8M + 4Q + 10 \tag{2.4}$$

- при нечётном N:

$$f_{opt} = f_I + f_{II} + f_{III} = 10MNQ + \frac{11}{2}MN + \frac{11}{2}NQ + 10NQ + 8M + 4Q + 6 \tag{2.5}$$

2.3 Список оптимизаций алгоритма Винограда

Алгоритм Винограда был оптимизирован с помощью следующих модификаций:

1. Замена конструкций вида a = a + b (трудоёмкость = 2) на a += b (трудоёмкость = 1).

2. Вычисление повторно используемых величин выполняется предварительно ($N/2$, $N-1$, $2*k$).

3. Цикл вычислений для нечётных элементов включён в основной цикл, добавив дополнительные операции. Данная модификация исключает повторные проверки на нечётность N.

2.4 Замер используемой памяти

Пусть даны две матрицы A и B размерностью $M \times N$ и размерностью $N \times Q$ соответственно и для хранения целого числа требуется 4 байта памяти.
В каждом из алгоритмов требуется хранить исходные матрицы A и B и матрицу результата умножения C, которая имеет размеры $M \times Q$. Таким образом, под хранение матриц требуется $4 \cdot (MN + NQ + MQ)$ байт памяти.
Для алгоритма Винограда и его оптимизированного варианта требуется также хранить два дополнительных вектора $MulH$ и $MulV$. Их размеры M и Q соответственно, следовательно требуется дополнительно $4 \cdot (M + Q)$ байт памяти. В итоге для алгоритмов Винограда требуется $4 \cdot (MN + NQ + MQ + M + Q)$ байт памяти. Таким образом, при больших размерах матриц (больше 100) алгоритмы Винограда будут значительно проигрывать по памяти классическому алгоритму.

3. Технологическая часть

3.1 Требования к программному обеспечению

На вход подаются размеры двух матриц, затем вводятся сами матрицы. На выход программа выдаёт три матрицы, которые являются результатами работы трёх различных алгоритмов умножения.

3.2 Средства реализации

Для реализации программы был использован язык C++ [1]. Для замера процессорного времени была использована функция rdtsc() из библиотеки stdrin.h.

3.3 Реализации алгоритмов

В листингах 3.1, 3.2, 3.3 представлены коды реализации алгоритмов умножения матриц.

Листинг 3.1: Классический алгоритм

```
1  void multiply_classic(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
2  {
3    for (unsigned i = 0; i < M; i++)
4      for (unsigned j = 0; j < Q; j++)
5      {
6        C[i][j] = 0;
7        for (unsigned k = 0; k < N; k++)
8          C[i][j] += A[i][k] * B[k][j];
9      }
10 }
```

Листинг 3.2: Алгоритм Винограда

```
1  void multiply_vinograd(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
2  {
3    int *MulH = new int[M];
4    for (unsigned i = 0; i < M; i++)
5    {
6      MulH[i] = 0;
7      for (unsigned k = 0; k < N / 2; k++)
8        MulH[i] = MulH[i] + A[i][2 * k] * A[i][2 * k + 1];
9    }
10
11   int *MulV = new int[Q];
12   for (unsigned i = 0; i < Q; i++)
13   {
```

```
    MulV[i] = 0;
    for (unsigned k = 0; k < N / 2; k++)
      MulV[i] = MulV[i] + B[2 * k][i] * B[2 * k + 1][i];
  }

  for (unsigned i = 0; i < M; i++)
    for (unsigned j = 0; j < Q; j++)
    {
      C[i][j] = -MulH[i] - MulV[j];
      for (unsigned k = 0; k < N / 2; k++)
        C[i][j] = C[i][j] + (A[i][2 * k] + B[2 * k + 1][j]) *
              (A[i][2 * k + 1] + B[2 * k][j]);

    }

  if (N % 2 == 1)
    for (unsigned i = 0; i < M; i++)
      for (unsigned j = 0; j < Q; j++)
        C[i][j] = C[i][j] + A[i][N - 1] * B[N - 1][j];

  delete [] MulH;
  delete [] MulV;
}
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
void multiply_vinograd_opt(int **A, int **B, int **C, unsigned M, unsigned N, unsigned Q)
{
  unsigned half_N = N >> 1;

  int *MulH = new int[M];
  for (unsigned i = 0; i < M; i++)
  {
    MulH[i] = 0;
    for (unsigned k = 0; k < half_N; k++)
    {
      k <<= 1;
      MulH[i] += A[i][k] * A[i][k + 1];
    }
  }

  int *MulV = new int[Q];
  for (unsigned i = 0; i < Q; i++)
  {
    MulV[i] = 0;
    for (unsigned k = 0; k < half_N; k++)
    {
      k <<= 1;
      MulV[i] += B[k][i] * B[k + 1][i];
    }
  }

  if (N % 2)
  {
```

```
29      unsigned N_minus_1 = N - 1;
30      for (unsigned i = 0; i < M; i++)
31        for (unsigned j = 0; j < Q; j++)
32        {
33          C[i][j] = A[i][N_minus_1] * B[N_minus_1][j] - MulH[i] - MulV[j];
34          for (unsigned k = 0; k < half_N; k++)
35          {
36            k <<= 1;
37            C[i][j] += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
38          }
39        }
40    }
41    else
42    {
43      for (unsigned i = 0; i < M; i++)
44        for (unsigned j = 0; j < Q; j++)
45        {
46          C[i][j] = -MulH[i] - MulV[j];
47          for (unsigned k = 0; k < half_N; k++)
48          {
49            k <<= 1;
50            C[i][j] += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
51          }
52        }
53    }
54
55    delete [] MulH;
56    delete [] MulV;
57  }
```

3.4 Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице 3.1. Входные данные удовлетворяют условиям, необходимым для умножения матриц, так как проверка на соответствие их размеров возложена на другую функцию.

Таблица 3.1: Функциональные тесты

Матрица 1	**Матрица 2**	**Ожидание**
$\begin{bmatrix} 5 \end{bmatrix}$	$\begin{bmatrix} -8 \end{bmatrix}$	$\begin{bmatrix} -40 \end{bmatrix}$
$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ -1 \\ 5 \end{bmatrix}$	$\begin{bmatrix} 6 \end{bmatrix}$
$\begin{bmatrix} 5 & 1 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 3 & -5 \\ 10 & 0 \end{bmatrix}$	$\begin{bmatrix} -10 & 25 \\ -10 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 0 \\ 0 & -2 \end{bmatrix}$	$\begin{bmatrix} 7 & 2 \\ 3 & 8 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 & -1 \\ 5 & -3 & -4 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix}$

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

4. Экспериментальная часть

4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы, демонстрирующий корректную работу алгоритмов.

Рис. 4.1: Пример работы программы

4.2 Сравнение работы алгоритмов при чётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 100 до 1000 с шагом 100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.2.

Таблица 4.1: Время работы алгоритмов при чётных размерах матриц в тактах процессора

Размер матриц	Классический	Алг-м Виногада	Оптимиз. алг-м Винограда
100	11378576	8801206	1183784
200	94712580	77139807	6132265
300	342637546	273553296	13960164
400	863174672	684590971	24498851
500	1792773181	1404195438	38730481
600	3521402245	2810464339	65344034
700	5996792976	4687597625	90821373
800	10493249242	8357682976	125045114
900	14519644100	11520503989	150466748
1000	23147499368	18975832272	182725943

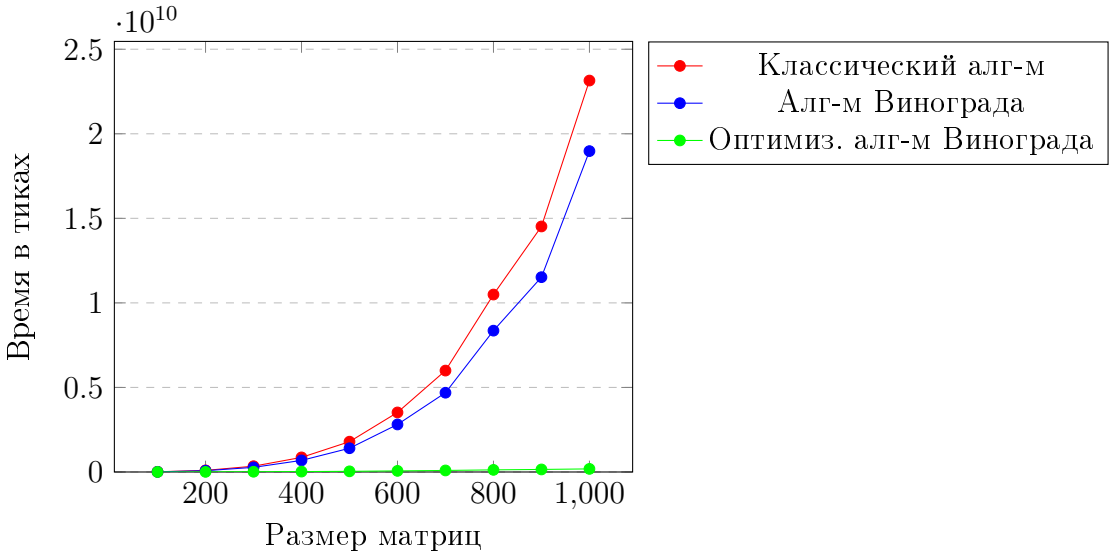

Рис. 4.2: График времени работы алгоритмов при чётных размерах матриц

Из результатов экспериментов можно сделать вывод о том, что алгоритм Винограда выигрывает классический алгоритм умножения матриц в среднем на 18%. Оптимизированный алгоритм имеет большой выигрыш во времени работы: его реализация работает быстрее в среднем в 10 раз, тем самым являясь самым эффективным по времени из трёх представленных.

4.3 Сравнение работы алгоритмов при нечётных размерах матрицы

Для сравнения времени работы алгоритмов умножения матриц были использованы квадратные матрицы размером от 101 до 1001 с шагом 100. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунке 4.3.

Таблица 4.2: Время работы алгоритмов при нечётных размерах матриц в тактах процессора

Размер матриц	Классический	Алг-м Виногада	Оптимиз. алг-м Винограда
101	11889847	9424292	1314911
201	97600814	77692429	5905464
301	356044760	286028727	15421621
401	939094754	750760519	27668312
501	2032880488	1606779606	42609865
601	4139224911	3146415361	71468977
701	6253434863	4930691171	95735554
801	9771180988	7753844038	122309346
901	14620168149	11646686358	156299495
1001	23445735293	19176489090	193180038

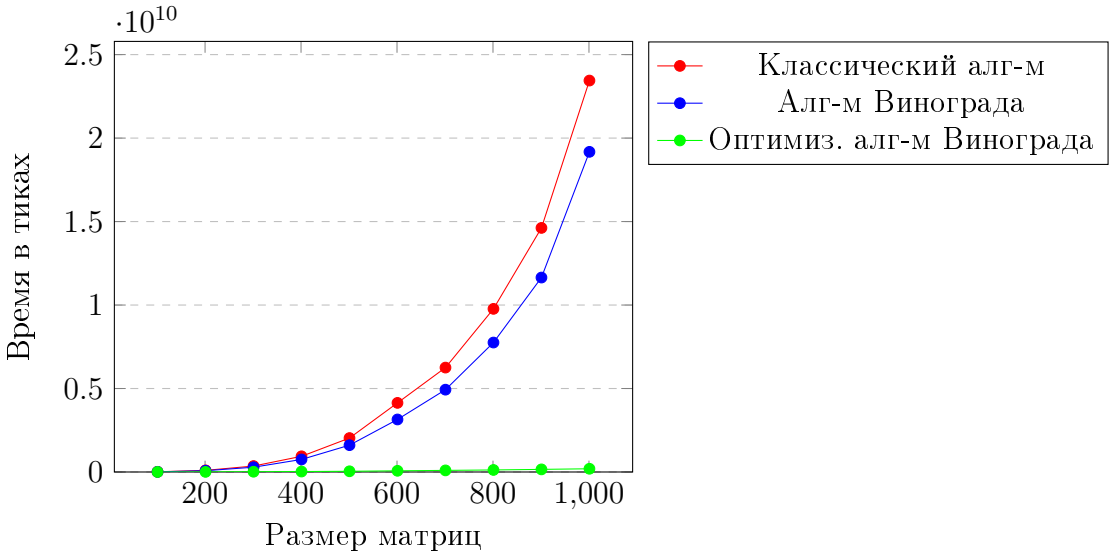

Рис. 4.3: График времени работы алгоритмов при нечётных размерах матриц

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 значительно 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 и увеличив тем самым расход памяти, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный вариант работает на 18% быстрее классического алгоритма, оптимизированный - в 10 раз.

Литература

[1] https://cppreference.com/ [Электронный ресурс]