Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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»»

Лабораторная работа № 1
по дисциплине «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Куликов Н.В.

Группа ИУ7-53Б

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

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы было исследование алгоритмов умножения матриц: стандартного, Винограда и оптимизированного Винограда.

Для достижения поставленной цели необходимо было выполнить следующие задачи:

1) описать и реализовать алгоритмы: стандартный и Винограда,
2) оптимизировать алгоритм Винограда,
3) оценить трудоёмкости алгоритмов,
4) сравнить время работы реализации стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда.

1 Аналитическая часть

1.1 Матрица

Матрицей размера $[m \times n]$ называют прямоугольную таблицу чисел, функций или алгебраических выражений, содержащую m строк и n столбцов. [1]

Пусть A — матрица, состоящая из m строк и n столбцов. Тогда она имеет вид:

$$A_{m \times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

где a_{ij} – элемент матрицы A, который находится на строке i и в столбце j.

Можно выделить следующие операции над матрицами:

1) сложение матриц одинаковых размеров,
2) вычитание матриц одинаковых размеров,
3) умножение матриц, если количество столбцов первой матрицы равно количеству строк второй матрицы. [1]

1.2 Стандартный алгоритм

Пусть даны прямоугольные матрицы $A_{m \times n}$ и $B_{n \times k}$:

$$A_{m \times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{l2} & \dots & a_{mn} \end{pmatrix}, \quad B_{n \times k} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{m2} & \dots & b_{nk} \end{pmatrix}, \tag{1.2}$$

Тогда матрица $C_{m \times k}$:

$$C_{m\times k} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{pmatrix}, \tag{1.3}$$

где

$$c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} \quad (i = \overline{1,m}; j = \overline{1,k}) \tag{1.4}$$

называется произведением матриц A и B.

1.3 Алгоритм Винограда

Алгоритм Винограда – алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. В 2020 году Джош Алман и Вирджиния Вильямс улучшили метод, достигнув оценки $O(n^{2,3728596})$. [2]

Два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$. Преобразовав выражение, получим:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \tag{1.5}$$

При нечётной длине перемножаемых векторов к каждой ячейке результирующей матрицы добавится одно слагаемое.

Свяжем стандартный метод умножения матриц со скалярным произведением строки матрицы A на столбец B. Скалярное произведение (1.5) можно произвести иначе:

$$C_{ij} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j} \tag{1.6}$$

Виноград предложил находить второе и третье слагаемые в формуле (1.5) на предварительном этапе вычислений, заранее для каждой строки матрицы A

и столбца B соответственно. Так, вычислив единожды для строки i матрицы A значение выражения

$$\sum_{k=1}^{n/2}(a_{i,2k-1}a_{i,2k}) \tag{1.7}$$

его можно далее использовать m раз при нахождении элементов строчки i матрицы C. Ведь данное выражение участвует в определении c_{ij} по формуле (1.6) для выбранного i и $1 \leq j \leq m$.

Аналогично, вычислив единожды для столбца j матрицы B значение выражения

$$\sum_{k=1}^{n/2}(b_{2k-1,j}b_{2k,j}) \tag{1.8}$$

его можно далее использовать n раз при нахождении элементов столбца j матрицы C. Ведь данное выражение участвует в определении c_{ij} по формуле (1.6) для выбранного j и $1 \leq i \leq n$. [2]

Это означает, что над предварительно обработанными элементами по формуле (1.5) придется выполнять лишь первые два умножения, последующие пять сложений и два дополнительных сложения. Операция сложения выполняется быстрее, поэтому на реализация алгоритма должна работать быстрее стандартного алгоритма перемножения матриц.

1.4 Оптимизированный алгоритм Винограда

Алгоритм Винограда оптимизирован следующим образом:

1) использовано предварительное вычисление значений (N-1, N/2 и др.),
2) использовано накопление в буфер значения при заполнении массивов и матрицы.

1.5 Вывод

В аналитической части были рассмотрены алгоритмы умножения матриц: стандартный алгоритм и алгоритм Винограда. Также были рассмотрены возможные оптимизации алгоритма Винограда.

2 Конструкторская часть

2.1 Функциональные требования

Разработать программное обеспечение с двумя режимами работы: одиночного выполнения умножения матриц и массированного замера времени выполнения реализаций алгоритмов умножения матриц при варьируемом линейном размере квадратных перемножаемых матриц.

Входные данные:

— пункт меню (целое число от 0 до 4),

— размеры матрицы,

— элементы матрицы.

Выходные данные:

Результат умножения матриц и результаты замеров процессорного времени выполнения реализаций алгоритмов умножения матриц.

2.2 Схемы алгоритмов

На рисунках 2.1 – 2.8 представлены схемы алгоритмов умножения матриц.

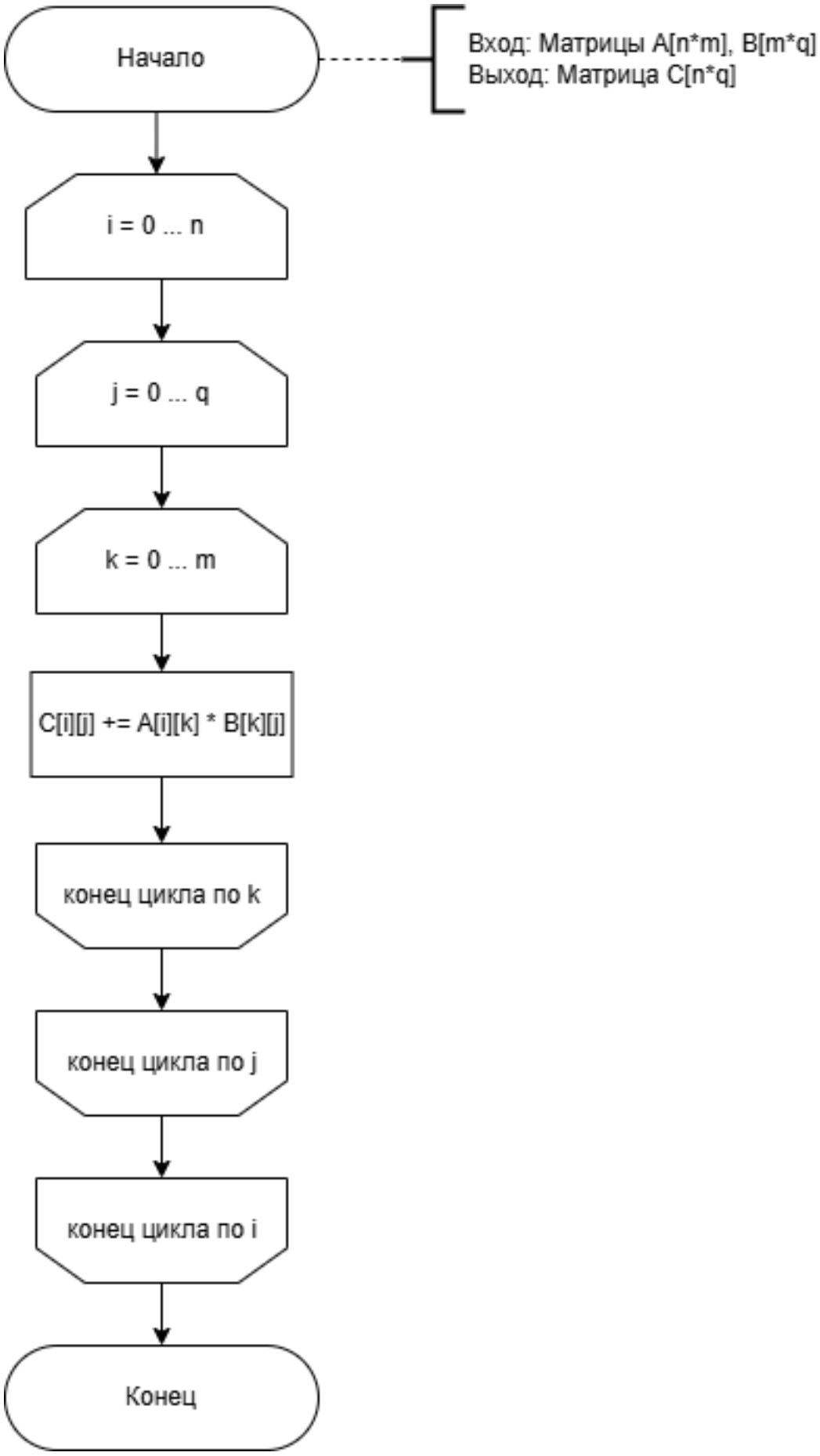

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

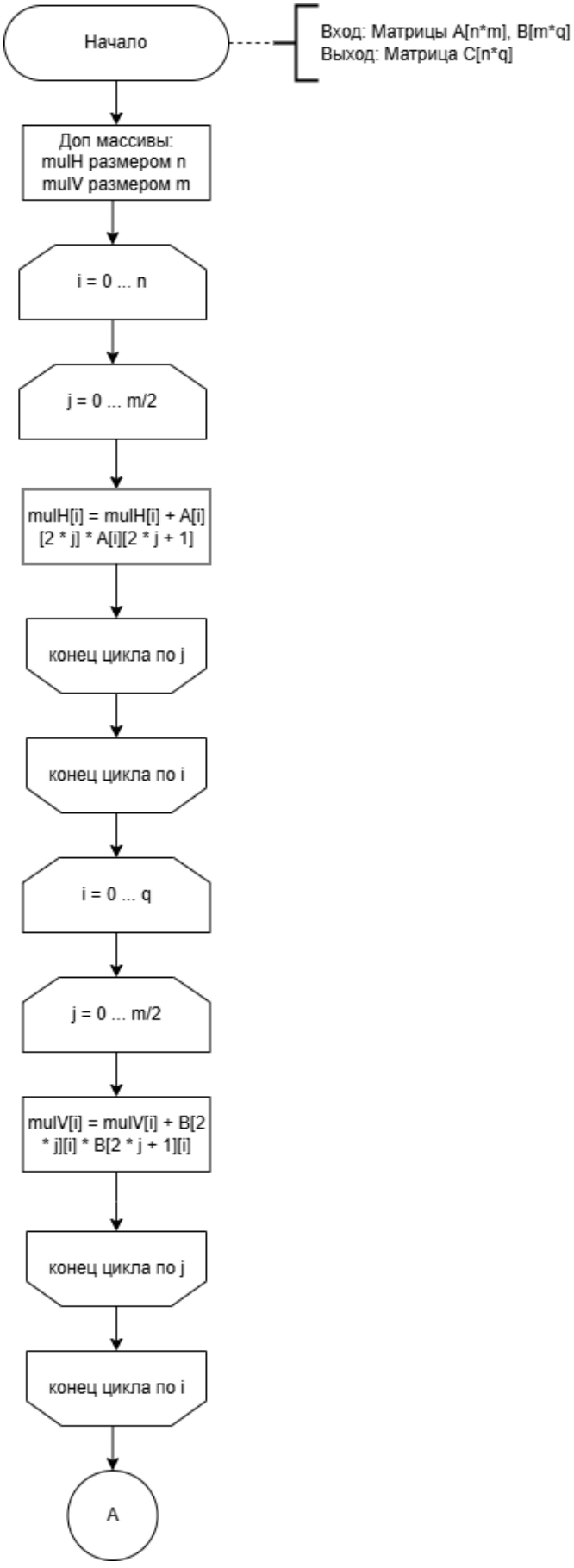

Рисунок 2.2 — Схема алгоритма Винограда (часть 1)

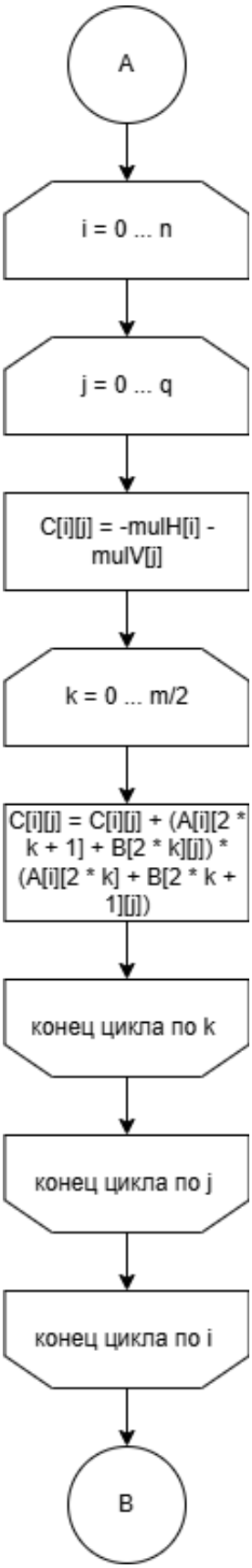

Рисунок 2.3 — Схема алгоритма Винограда (часть 2)

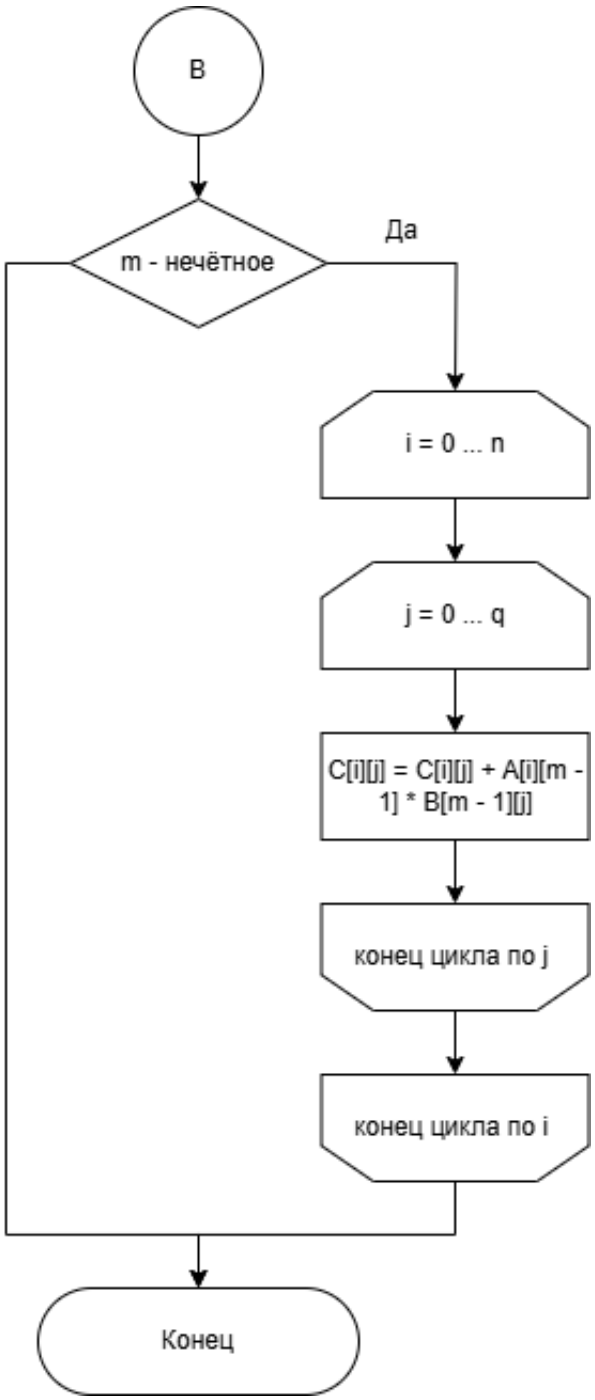

Рисунок 2.4 — Схема алгоритма Винограда (часть 3)

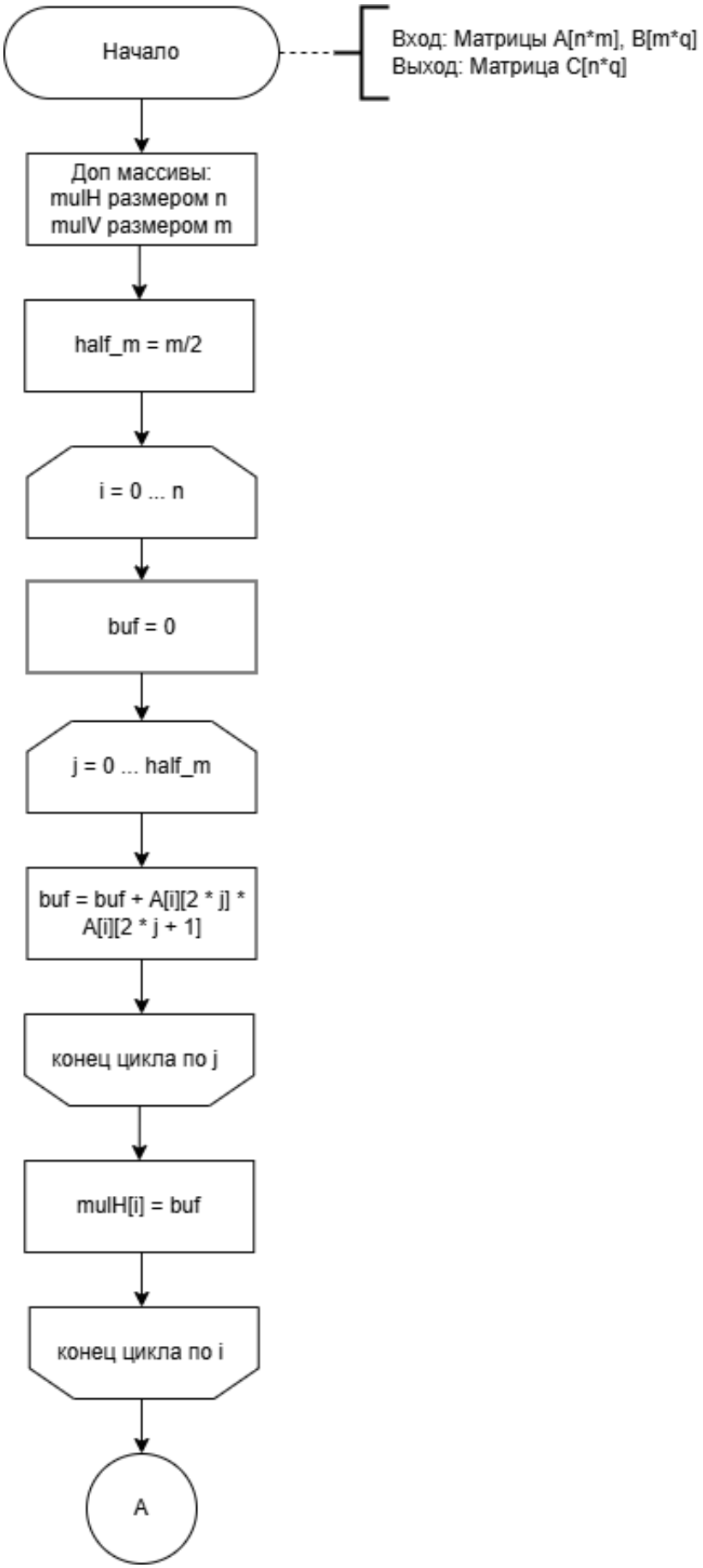

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда (часть 1)

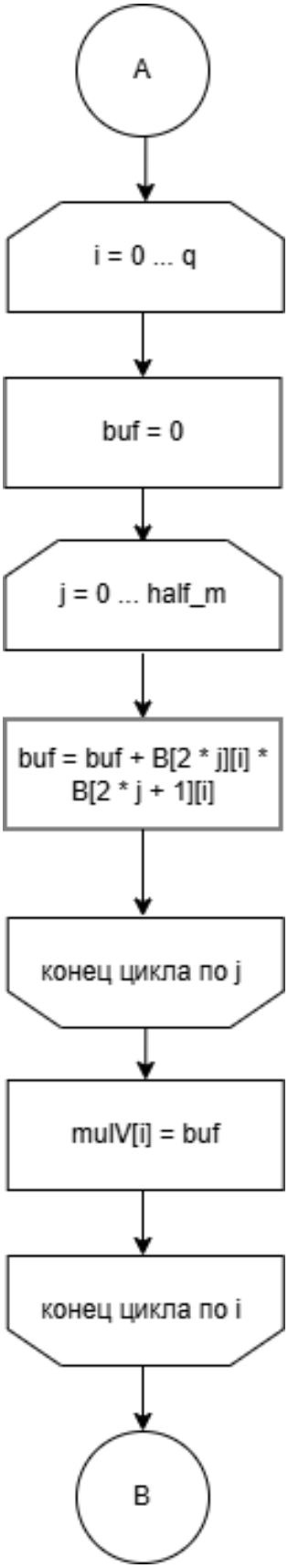

Рисунок 2.6 — Схема оптимизированного алгоритма Винограда (часть 2)

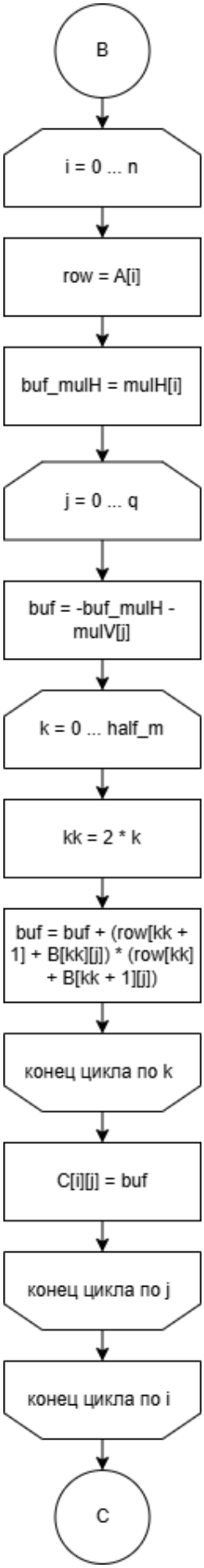

Рисунок 2.7 — Схема оптимизированного алгоритма Винограда (часть 3)

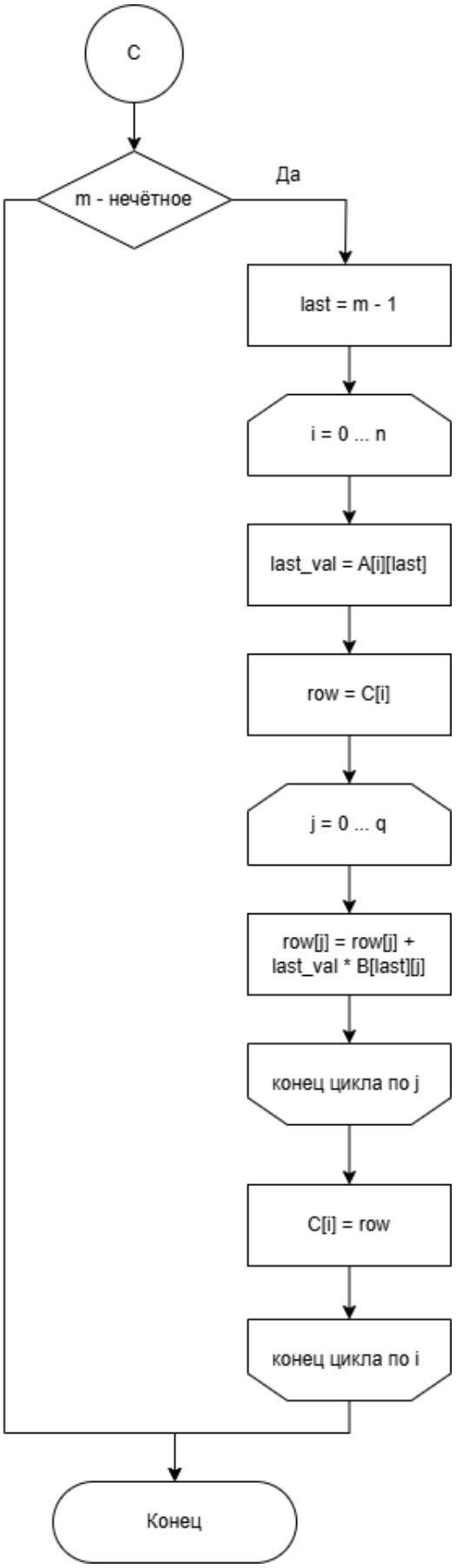

Рисунок 2.8 — Схема оптимизированного алгоритма Винограда (часть 4)

2.3 Модель вычислений

Чтобы вычислить трудоёмкость алгоритмов, введена следующая модель вычислений:

1) базовые операции:
   — трудоёмкость операций из списка (2.1) равна 1:

$$=, +, +=, -, -=, ++, --, ==, !=, <, <=, >=, >, [], \&\&, \\ \&, >>, <<, ||, | \tag{2.1}$$

   — трудоёмкость операций из списка (2.2) равна 2:

$$*, *=, /, /=, \%, \%= \tag{2.2}$$

2) трудоёмкость условного перехода равна 0,
3) трудоёмкость условного оператора по формуле (2.3):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{– лучший случай} \\ max(f_A, f_B) & \text{– худший случай} \end{cases} \tag{2.3}$$

4) трудоёмкость цикла по формуле (2.4):

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

где M – число итераций

2.4 Трудоёмкость алгоритмов

2.4.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма рассчитана по формулам (2.5 – 2.9):

$$f_{firstCycle} = 2 + N \cdot (2 + f_{secondCycle}) \tag{2.5}$$

$$f_{secondCycle} = 2 + Q \cdot (2 + f_{thirdCycle}) \tag{2.6}$$

$$f_{thirdCycle} = 2 + M \cdot (2 + f_{body}) \tag{2.7}$$

$$f_{body} = 1 + 8 + 1 + 2 = 12 \tag{2.8}$$

$$f_{algo} = 2+N\cdot(2+2+Q\cdot(2+2+M\cdot(2+12))) = 14NQM+4NQ+4N+2 \tag{2.9}$$

Асимптотическая оценка временной сложности алгоритма: $O(14NQM)$. В частном случае квадратных матриц с линейной размерностью N: $O(N^3)$

2.4.2 Алгоритм Винограда

— трудоёмкость инициализации массивов mulH и mulV рассчитана по формуле (2.10):

$$f_{init} = N + M \tag{2.10}$$

— трудоёмкость вычисления mulH рассчитана по формулам (2.11) – (2.14):

$$f_{firstCycle} = 2 + N \cdot (2 + f_{secondCycle}) \tag{2.11}$$

$$f_{secondCycle} = 4 + \frac{M}{2} \cdot (4 + f_{body}) \tag{2.12}$$

$$f_{body} = 1 + 6 + 1 + 2 + 5 = 15 \tag{2.13}$$

$$f_{mulH} = 2 + N \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 15)) = \frac{19}{2}NM + 6N + 2 \tag{2.14}$$

— трудоёмкость вычисления mulV рассчитана по формулам (2.15) – (2.18):

$$f_{firstCycle} = 2 + N \cdot (2 + f_{secondCycle}) \tag{2.15}$$

$$f_{secondCycle} = 4 + M/2 \cdot (4 + f_{body}) \tag{2.16}$$

$$f_{body} = 1 + 6 + 1 + 2 + 5 = 15 \tag{2.17}$$

$$f_{mulV} = 2 + N \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 15)) = \frac{19}{2}NM + 6N + 2 \tag{2.18}$$

— трудоёмкость главного цикла рассчитана по формулам (2.19) – (2.23):

$$f_{firstCycle} = 2 + N \cdot (2 + f_{secondCycle}) \tag{2.19}$$

$$f_{secondCycle} = 2 + Q \cdot (2 + 7 + f_{thirdCycle}) \tag{2.20}$$

$$f_{thirdCycle} = 4 + \frac{M}{2} \cdot (4 + f_{body}) \tag{2.21}$$

$$f_{body} = 12 + 3 + 2 + 1 + 10 = 28 \tag{2.22}$$

$$\begin{gathered} f_{cycle} = 2 + N \cdot (2 + 2 + Q \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28))) = \\ = 16NQM + 13NQ + 4N + 2 \end{gathered} \tag{2.23}$$

— трудоёмкость цикла, нужного для подсчёта значений при нечётном размере матрицы, расчитана по формуле (2.24):

$$f_{odd} = 3 + \begin{cases} 2 + N \cdot (2 + 2 + Q \cdot (2 + 14)), & \text{– нечётный размер} \\ 0, & \text{– иначе} \end{cases} \tag{2.24}$$

Трудоёмкость алгоритма Винограда для худшего случая, когда у матрицы нечётный размер, рассчитана по формуле (2.25):

$$\begin{gathered} f_{algoWorst} = f_{init} + f_{mulH} + f_{mulV} + fcycle + f_{odd} = \\ = N + M + \frac{19}{2}NM + 6N + 2 + \frac{19}{2}NM + 6N + 2 + 16NQM + \\ + 13NQ + 4N + 2 + 3 + 2 + N \cdot (2 + 2 + Q \cdot (2 + 14)) = \\ = 16NQM + 29NQ + 19NM + 21N + M + 11 \end{gathered} \tag{2.25}$$

Трудоёмкость алгоритма Винограда для лучшего случая, когда у матрицы чётный размер, рассчитана по формуле (2.26):

$$\begin{gathered} f_{algoBest} = f_{init} + f_{mulH} + f_{mulV} + fcycle + f_{odd} = \\ = N + M + \frac{19}{2}NM + 6N + 2 + \frac{19}{2}NM + 6N + 2 + 16NQM + \\ + 13NQ + 4N + 2 + 3 = 16NQM + 13NQ + 19NM + 17N + M + 9 \end{gathered} \tag{2.26}$$

Асимптотическая оценка временной сложности алгоритма для худшего и лучшего случая: $O(16NQM)$. В частном случае квадратных матриц с линейной размерностью N: $O(N^3)$

2.4.3 Оптимизированный алгоритм Винограда

— трудоёмкость инициализации массивов mulH и mulV рассчитана по формуле (2.27):

$$f_{init} = N + M \tag{2.27}$$

— трудоёмкость предварительного вычисления $\frac{M}{2}$ равна 3,

— трудоёмкость вычисления mulH рассчитана по формулам (2.28) – (2.31):

$$f_{firstCycle} = 2 + N \cdot (5 + f_{secondCycle}) \tag{2.28}$$

$$f_{secondCycle} = 2 + \frac{M}{2} \cdot (2 + f_{body}) \tag{2.29}$$

$$f_{body} = 1 + 4 + 1 + 2 + 5 = 13 \tag{2.30}$$

$$f_{mulH} = 2 + N \cdot (5 + 2 + \frac{M}{2} \cdot (2 + 13)) = \frac{15}{2}NM + 7N + 2 \tag{2.31}$$

— трудоёмкость вычисления mulV рассчитана по формулам (2.32 – (2.35):

$$f_{firstCycle} = 2 + N \cdot (5 + f_{secondCycle}) \tag{2.32}$$

$$f_{secondCycle} = 2 + \frac{M}{2} \cdot (2 + f_{body}) \tag{2.33}$$

$$f_{body} = 1 + 4 + 1 + 2 + 5 = 13 \tag{2.34}$$

$$f_{mulV} = 2 + N \cdot (5 + 2 + \frac{M}{2} \cdot (2 + 13)) = \frac{15}{2}NM + 7N + 2 \tag{2.35}$$

— трудоёмкость главного цикла рассчитана по формулам (2.36 – (2.40):

$$f_{firstCycle} = 2 + N \cdot (2 + 4 + f_{secondCycle}) \tag{2.36}$$

$$f_{secondCycle} = 2 + Q \cdot (2 + 7 + f_{thirdCycle}) \tag{2.37}$$

$$f_{thirdCycle} = 2 + \frac{M}{2} \cdot (2 + 3 + f_{body}) \tag{2.38}$$

$$f_{body} = 4 + 2 + 6 + 2 = 14 \tag{2.39}$$

$$\begin{gathered} f_{cycle} = 2 + N \cdot (6 + 2 + Q \cdot (2 + 7 + 2 + \frac{M}{2} \cdot (5 + 14))) = \\ = \frac{19}{2}NQM + 11NQ + 8N + 2 \end{gathered} \tag{2.40}$$

— трудоёмкость цикла, нужного для подсчёта значений при нечётном

размере матрицы, расчитана по формуле (2.41):

$$f_{odd} = 3 + \begin{cases} 4 + N \cdot (2 + 7 + 2 + Q \cdot (2 + 8)), & \text{– нечётный размер} \\ 0, & \text{– иначе} \end{cases} \tag{2.41}$$

Трудоёмкость оптимизированного алгоритма Винограда для худшего случая, когда у матрицы нечётный размер, рассчитана по формуле (2.42):

$$\begin{gathered} f_{algoWorst} = f_{init} + f_{mulH} + f_{mulV} + fcycle + f_{odd} = \\ = N + M + \frac{15}{2}NM + 7N + 2 + \frac{15}{2}NM + 7N + 2 + \frac{19}{2}NQM + \\ + 11NQ + 8N + 2 + 3 + 4 + N \cdot (2 + 7 + 2 + Q \cdot (2 + 8)) = \\ = \frac{19}{2}NQM + 21NQ + 15NM + 34N + M + 13 \end{gathered} \tag{2.42}$$

Трудоёмкость оптимизированного алгоритма Винограда для лучшего случая, когда у матрицы чётный размер, рассчитана по формуле (2.43):

$$\begin{gathered} f_{algoBest} = f_{init} + f_{mulH} + f_{mulV} + fcycle + f_{odd} = \\ = N + M + \frac{15}{2}NM + 7N + 2 + \frac{15}{2}NM + 7N + 2 + \frac{19}{2}NQM + \\ + 11NQ + 8N + 2 + 3 = \frac{19}{2}NQM + 11NQ + 15NM + 23N + M + 9 \end{gathered} \tag{2.43}$$

Асимптотическая оценка временной сложности алгоритма для худшего и лучшего случая: $O(\frac{19}{2}NQM)$. В частном случае квадратных матриц с линейной размерностью N: $O(N^3)$

2.5 Вывод

В данном разделе были описаны функциональные требования к программе, построены схемы алгоритмов умножения матриц: стандартного, алгоритма Винограда и оптимизированного алгоритма Винограда.

Была введена модель вычислений, в соответствии с которой были рассчитаны трудоёмкости алгоритмов умножения матриц. Проведённая оценка трудоемкости алгоритмов показала, что трудоёмкость выполнения алгоритма Винограда в случае оптимизации в 1.68 раз меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выьран язык C. Выбор обусловлен тем, что C – статически типизированный язык программирования, в нём нет сборщика мусора и имеется стандартная библиотека для замера процессорного времени.

Для замера процессорного времени использовалась функция clock() из модуля time. [3]

3.2 Реализация алгоритмов

В листингах (3.1 - 3.3) показаны реализации алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

```
void multiply_matrix_standart_algo(matrix_t *res, matrix_t mtr1,
                                   matrix_t mtr2) {
  size_t n = mtr1.n;
  size_t m = mtr1.m;
  size_t q = mtr2.m;
  int **a = mtr1.data;
  int **b = mtr2.data;
  int **c = res->data;

  for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < q; j++)
      for (size_t k = 0; k < m; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

Листинг 3.1 — Реализация стандартного алгоритма умножения матриц

```
void winograd_algo(matrix_t *res, matrix_t mtr1, matrix_t mtr2) {
  size_t n = mtr1.n;
  size_t m = mtr1.m;
  size_t q = mtr2.m;
  int **a = mtr1.data;
  int **b = mtr2.data;
  int **c = res->data;

  int *mulH = calloc(n, sizeof(int));
  int *mulV = calloc(m, sizeof(int));

  for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < m / 2; j++)
      mulH[i] = mulH[i] + a[i][2 * j] * a[i][2 * j + 1];

  for (size_t i = 0; i < q; i++)
    for (size_t j = 0; j < m / 2; j++)
      mulV[i] = mulV[i] + b[2 * j][i] * b[2 * j + 1][i];

  for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < q; j++) {
      c[i][j] = -mulH[i] - mulV[j];
      for (size_t k = 0; k < m / 2; k++)
        c[i][j] = c[i][j] + (a[i][2 * k + 1] + b[2 * k][j]) * (a[
           i][2 * k] + b[2 * k + 1][j]);
    }

  if (m % 2 == 1)
    for (size_t i = 0; i < n; i++)
      for (size_t j = 0; j < q; j++)
        c[i][j] = c[i][j] + a[i][m - 1] * b[m - 1][j];

  free(mulH);
  free(mulV);
}
```

Листинг 3.2 — Реализация алгоритма Винограда

```
void optimized_winograd_algo(matrix_t *res, matrix_t mtr1,
   matrix_t mtr2) {
  size_t n = mtr1.n;
  size_t m = mtr1.m;
  size_t q = mtr2.m;
  int **a = mtr1.data;
  int **b = mtr2.data;
  int **c = res->data;

  int *mulH = calloc(n, sizeof(int));
  int *mulV = calloc(m, sizeof(int));

  size_t half_m = m / 2;

  for (size_t i = 0; i < n; i++) {
    int buf = 0;
    for (size_t j = 0; j < half_m; j++)
      buf = buf + a[i][2 * j] * a[i][2 * j + 1];
    mulH[i] = buf;
  }

  for (size_t i = 0; i < q; i++) {
    int buf = 0;
    for (size_t j = 0; j < half_m; j++)
      buf = buf + b[2 * j][i] * b[2 * j + 1][i];
    mulV[i] = buf;
  }

  for (size_t i = 0; i < n; i++) {
    int *row = a[i];
    int buf_mulH = mulH[i];
    for (size_t j = 0; j < q; j++) {
      int buf = -buf_mulH - mulV[j];
      for (size_t k = 0; k < half_m; k++) {
        size_t kk = 2 * k;
        buf = buf + (row[kk + 1] + b[kk][j]) * (row[kk] + b[kk +
           1][j]);
      }
      c[i][j] = buf;
```

```
    }
  }

  if (m % 2 == 1) {
    size_t last = m - 1;
    for (size_t i = 0; i < n; i++) {
      int last_val = a[i][last];
      int *row = c[i];
      for (size_t j = 0; j < q; j++)
        row[j] = row[j] + last_val * b[last][j];
      c[i] = row;
    }
  }

  free(mulH);
  free(mulV);
}
```

Листинг 3.3 — Реализация оптимизированного алгоритма Винограда

3.3 Функциональные тесты

В таблице 3.1 представлены результаты функционального тестирования реализаций: стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Каждая реализация каждого алгоритма прошла тесты успешно.

Таблица 3.1 — Результаты функционального тестирования

Входные данные		**Результат**	
Матрица A	**Матрица B**	**Ожидаемый**	**Фактический**
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	()	Ошибка	Ошибка
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$(1, 2, 3)$	Ошибка	Ошибка
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{pmatrix}$	$\begin{pmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{pmatrix}$

3.4 Вывод

В этом разделе были описаны средства реализации алгоритмов. Также были продемонстрированы листинги реализаций: стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Приведены результаты функционального тестирования.

4 Исследовательская часть

4.1 Технические характеристики

Замеры процессорного времени проводились на ноутбуке ACER Predator со следующими техническими характеристиками:

— процессор: Intel(R) Core(TM) i7-10750H CPU @ 2.60ГГц 2.59 ГГц,

— ОЗУ: 16 ГБ,

— ОС: Windows 10 Pro 64-разрядная.

Во время замеров процессорного времени ноутбук был подключен к электропитанию, сторонними приложениями нагружен не был.

4.2 Замеры процессорного времени

Были проведены замеры времени работы реализаций алгоритмов для данных, соответствующих лучшему и худшему случаю по трудоёмкости алгоритма Винограда, соответственно когда матрицы на входе имеют чётный размер и нечётный.

Результаты замеров времени для лучшего случая трудоёмкости алгоритма Винограда приведены в таблице 4.1:

Таблица 4.1 — Результаты замеров времени для лучшего случая

Размер матрицы	Стандартный алгоритм, мс	Винограда алгоритм, мс	Оптимизированный алгоритм Винограда, мс
50	0.36	0.265	0.219
100	2.828	2.016	1.687
150	9.453	6.953	5.688
200	22.797	16.594	13.203
250	44.875	33.203	26.422
300	81.515	61.594	48.406
350	146.45	111.3	83.765
400	216.95	158.95	125.39

На рисунке 4.1 изображён график зависимости времени работы алгоритмов от размеров матрицы.

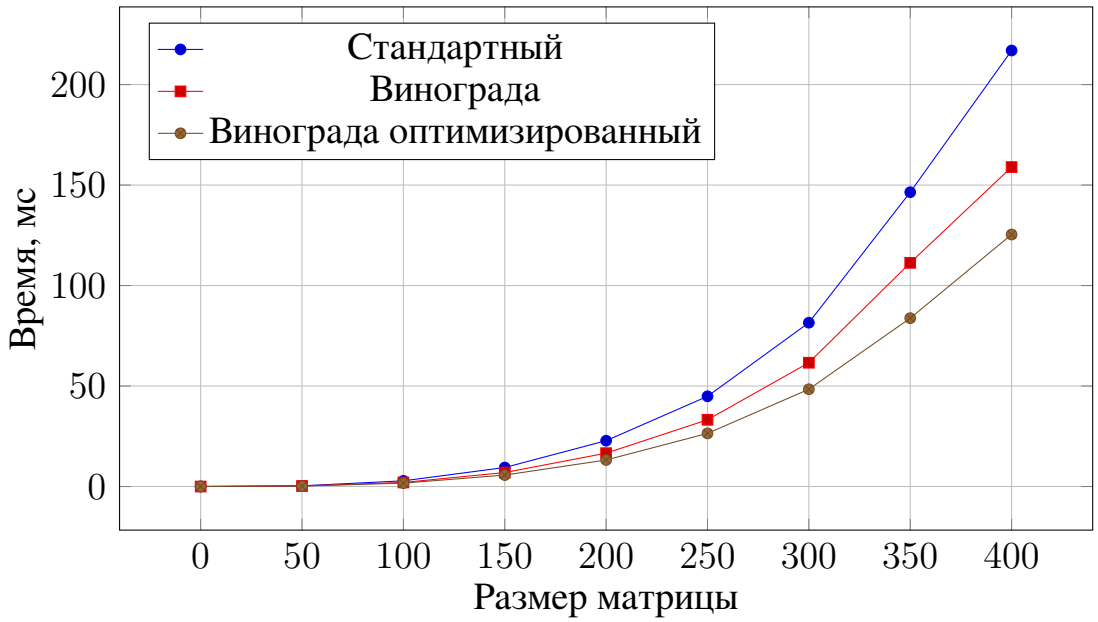

Рисунок 4.1 — Сравнение времени работы алгоритмов на чётных размерах матриц

Результаты замеров времени для худшего случая трудоёмкости алгоритма Винограда приведены в таблице 4.2:

Таблица 4.2 — Результаты замеров времени для худшего случая

Размер матрицы	Стандартный алгоритм, мс	Винограда алгоритм, мс	Оптимизированный алгоритм Винограда, мс
51	0.39	0.281	0.25
101	2.938	2.125	1.734
151	9.719	7.172	5.672
201	23.219	17.015	13.656
251	51.344	38.797	30.313
301	83.078	62.531	49.031
351	145.42	108.78	85.265
401	219.05	163.5	126.91

На рисунке 4.2 изображён график зависимости времени работы алгоритмов от размеров матрицы.

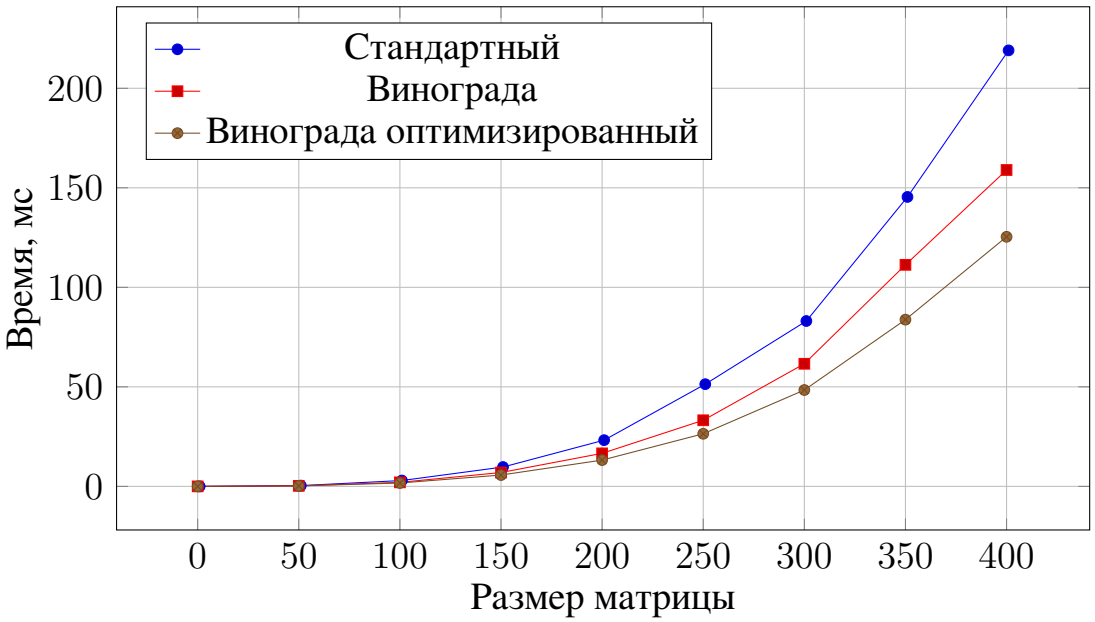

Рисунок 4.2 — Сравнение времени работы алгоритмов на нечётных размерах матриц

Из графиков 4.1 – 4.2 следует, что быстрее всех работает реализация оптимизированного алгоритма Винограда, а медленее всех стандартного алгоритма умножения матриц. Причём оптимизированный алгоритм Винограда быстрее стандартного примерно в 1,7 раз, а алгоритм Винограда без оптимизаций медленее оптимизированного в 1,2 раз. Также алгоритм Винограда на данных, соответствующих лучшему случаю, работает на 6% быстрее, чем на данных, соответствующих худшему случаю.

4.3 Вывод

В данном разделе были описаны технические характеристики машины, на которой проводились замеры времени. Продемонстрированы результаты замеров процессорного времени, был проведён сравнительный анализ времени работы алгоритмов умножения матриц.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были исследованы алгоритмы умножения матриц: стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

Выполнены задачи:

1) описаны и реализованы алгоритмы: стандартный и Винограда,
2) оптимизирован алгоритм Винограда,
3) оценены трудоёмкости алгоритмов,
4) сравнено время работы реализации стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда.

В результате лабораторной работы было выявлено, что оптимизированный алгоритм Винограда быстрее стандартного примерно в 1,7 раз, а алгоритм Винограда без оптимизаций медленее оптимизированного в 1,2 раз. Также алгоритм Винограда на данных, соответствующих лучшему случаю (когда размеры матриц чётные), работает на 6% быстрее, чем на данных, соответствующих худшему случаю (когда размеры матриц нечётные).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Белоусов И. В. Матрицы и определители: учебное пособие по линейной алгебре. — Кишинев, 2006. — С. 1–11.

2. Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. — Самара: Изд-во Самарского университета, 2019. — С. 28–35.

3. ISO/IEC 9899:1999. 2007. — С. 7.23.2.1.