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

Отчёт по лабораторной работе №2 по дисциплине "Анализ алгоритмов"

Тема Умножение матриц

Студент Жабин Д.В.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2021 г.

Оглавление

Введение **4**

1 Аналитическая часть **5**

1.1 Стандартный алгоритм умножения матриц 5

1.2. Алгоритм умножения матриц Копперсмита–Винограда 5

1.3 Вывод по аналитической части 6

2 Конструкторская часть **7**

2.1 Схемы алгоритмов 7

2.2 Модель вычислений трудоемкости алгоритмов 16

2.3 Трудоемкость алгоритмов умножения матриц 17

2.3.1 Стандартный алгоритм 17

2.3.2 Алгоритм Копперсмита–Винограда 18

2.3.3 Оптимизированный алгоритм Копперсмита–Винограда . . 18

2.4 Вывод по конструкторской части 19

3 Технологическая часть **20**

3.1 Требования к ПО 20

3.2 Средства реализации 20

3.3 Реализация алгоритмов 20

3.4 Тестовые данные 23

3.5 Вывод по технологической части 24

4 Исследовательская часть **25**

4.1 Технические характеристики 25

4.2 Время выполнения реализаций алгоритмов 25

4.3 Вывод по исследовательской части 28

Заключение **29**

Литература **30**

Введение

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Элементы новой матрицы получаются из элементов исходных матриц в соответствии с определенными правилами.

Матрицы A и B могут быть перемножены, если они совместимы — число столбцов матрицы A равно числу строк матрицы B.

Тогда как матрицы используются для описания, в частности, преобразований математических пространств (поворот, отражение, растяжение и др.), произведение матриц описывает композицию преобразований.

Существует множество алгоритмов умножения матриц. Один из них — алгоритм Копперсмита–Винограда [4]. Это 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. Алгоритм Копперсмита–Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

Так, целью лабораторной работы является исследование стандартного алгоритма умножения матриц, алгоритма Копперсмита–Винограда и оптимизированного алгоритма Копперсмита–Винограда. Для этого поставлены следующие задачи:

- изучить и реализовать вышеперечисленные алгоритмы умножения матриц;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных.

1 Аналитическая часть

Рассмотрим ключевые особенности выбранных для анализа алгоритмов умножения матриц.

1.1 Стандартный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы размерами $M \times N$ и $N \times Q$ соответственно.

$$A_{MN} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix}, \quad B_{NQ} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1Q} \\ b_{21} & b_{22} & \dots & b_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NQ} \end{pmatrix} \tag{1.1}$$

Тогда полученная в ходе умножения матрица C размером $M \times Q$ будет называться произведением матриц A и B.

$$C_{MQ} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1Q} \\ c_{21} & c_{22} & \dots & c_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M1} & c_{M2} & \dots & c_{MQ} \end{pmatrix} \tag{1.2}$$

В матрице C (1.2) $c[i][j]$ вычисляется по формуле (1.3).

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \quad (i = \overline{1, M}; j = \overline{1, Q}) \tag{1.3}$$

Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм умножения матриц Копперсмита–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 $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение (1.4).

$$V \times W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{1.4}$$

Это равенство можно переписать в виде (1.5).

$$V \times W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{1.5}$$

Кажется, что выражение (1.5) задает больше работы, чем выражение (1.4): вместо четырех умножений — их шесть, а вместо трех сложений — десять. Менее очевидно, что выражение в правой части равенства (1.5) допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Вывод по аналитической части

В данном разделе были рассмотрены алгоритмы умножения матриц: стандартный алгоритм и алгоритм Копперсмита–Винограда.

2 Конструкторская часть

На основе полученных аналитических данных построим схемы алгоритмов умножения матриц и оценим их трудоемкости.

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2 представлены схемы алгоритмов умножения матриц: стандартного и Копперсмита–Винограда соответственно.

Нетрудно заметить, что алгоритм на рисунке 2.2 можно оптимизировать для достижения более быстрой его работы. Проделаем следующее:

- заменим операции вида $\beta = \beta + \alpha$ на $\beta += \alpha$;
- заменим в циклах по k шаг на 2, тем самым избавимся от двух операций умножения на каждой итерации.

Таким образом, на рисунке 2.3 представлена схема оптимизированного алгоритма умножения матриц Копперсмита–Винограда.

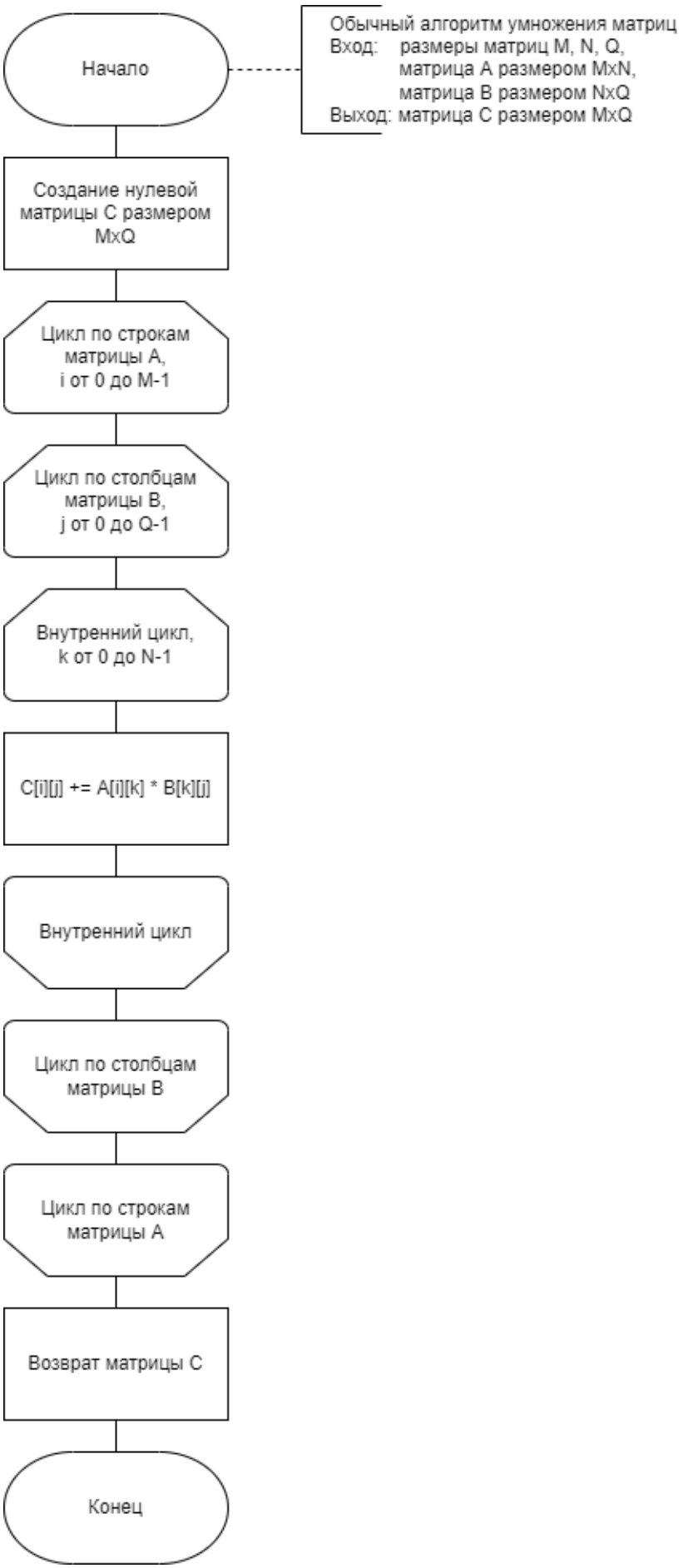
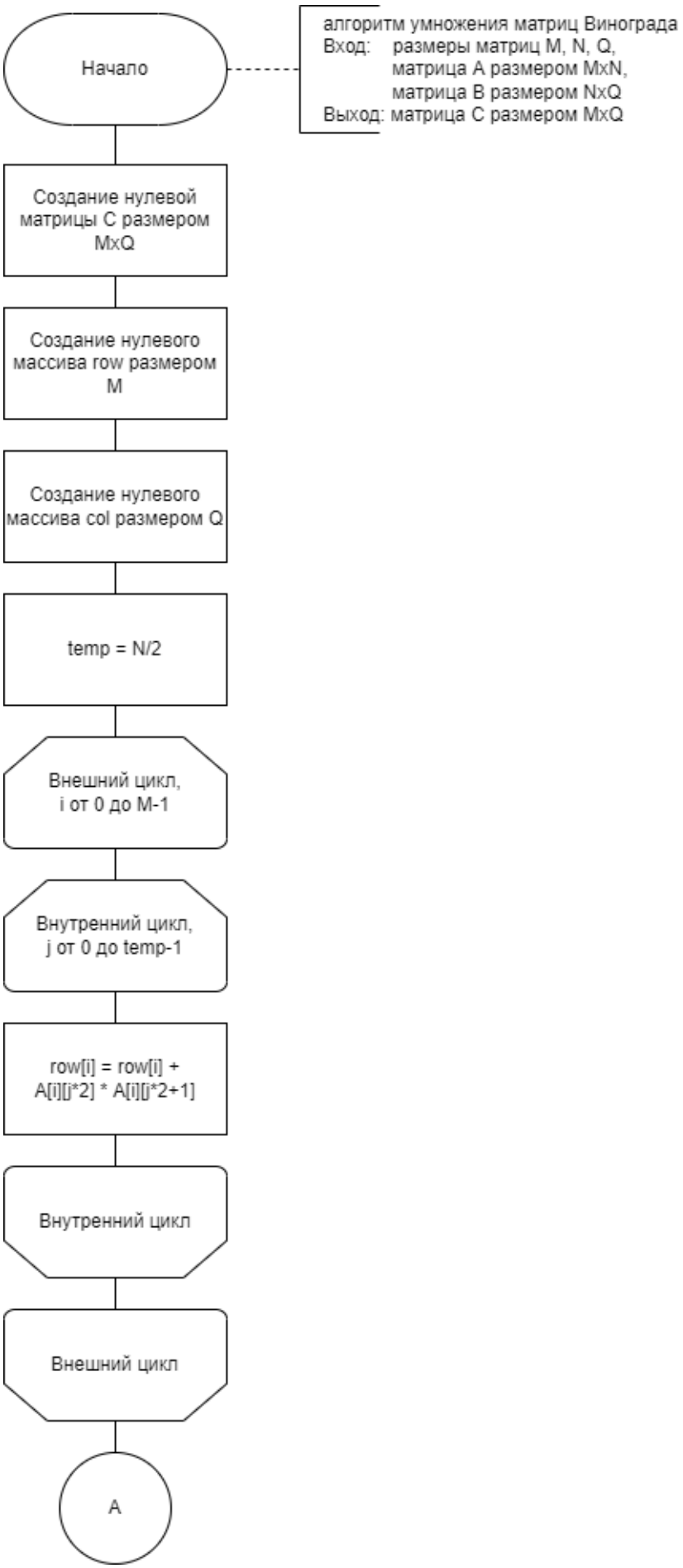
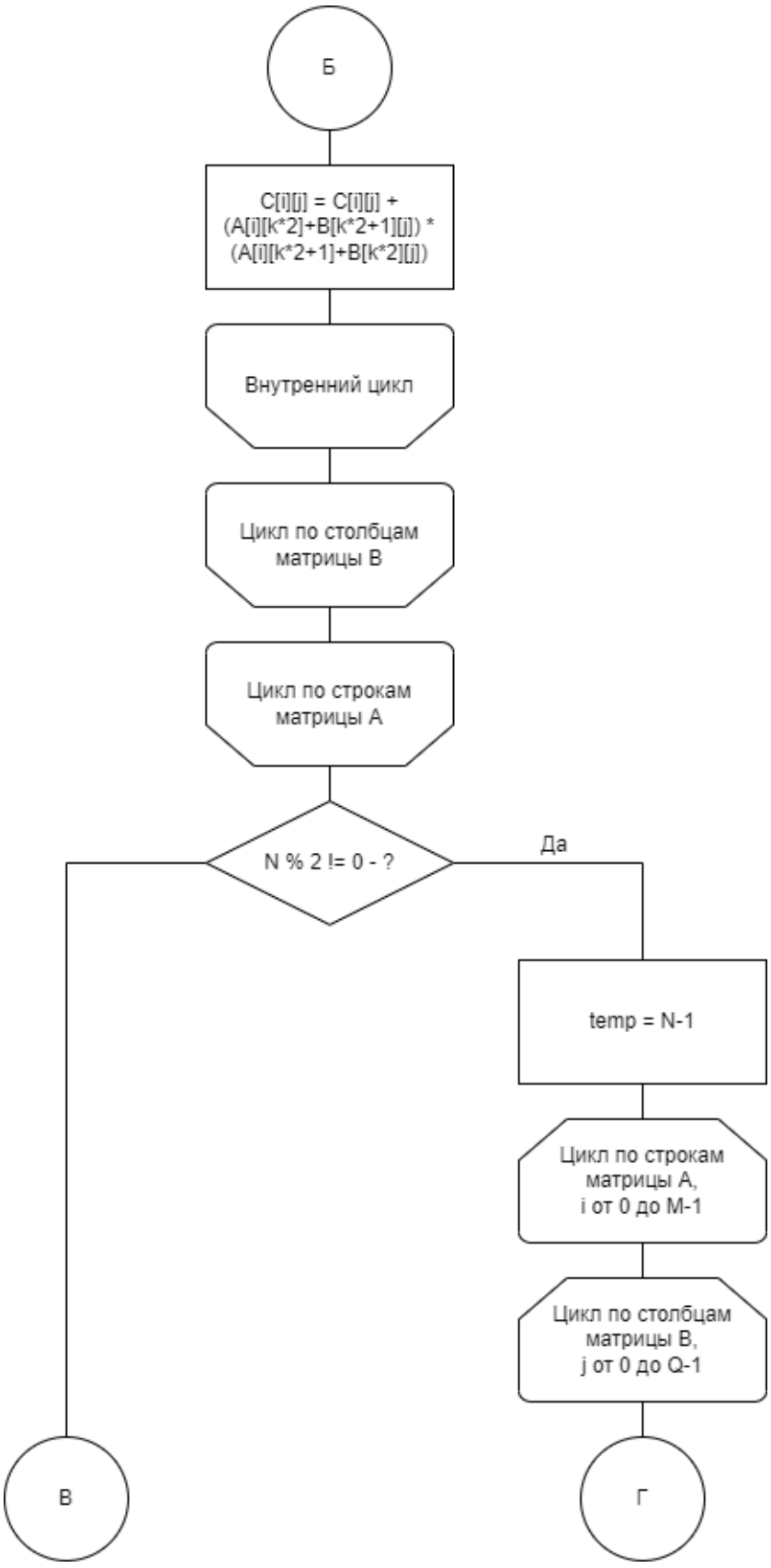


Рисунок 2.1 — Стандартный алгоритм умножения матриц


Начало
алгоритм умножения матриц Винограда
Вход: размеры матриц M, N, Q,
матрица A размером MxN,
матрица B размером NxQ
Выход: матрица C размером MxQ
Создание нулевой матрицы C размером MxQ
Создание нулевого массива row размером M
Создание нулевого массива col размером Q
temp = N/2
Внешний цикл, i от 0 до M-1
Внутренний цикл, j от 0 до temp-1
row[i] = row[i] + A[i][j*2] * A[i][j*2+1]
Внутренний цикл
Внешний цикл
A


A
Внешний цикл,
j от 0 до Q-1
Внутренний цикл,
i от 0 до temp-1
col[j] = col[j] +
B[i*2][j] * B[i*2+1][j]
Внутренний цикл
Внешний цикл
Цикл по строкам
матрицы A,
i от 0 до M-1
Цикл по столбцам
матрицы B,
j от 0 до Q-1
C[i][j] = -row[i]-col[j]
Внутренний цикл,
k от 0 до temp-1
Б


Б
C[i][j] = C[i][j] +
(A[i][k*2]+B[k*2+1][j]) *
(A[i][k*2+1]+B[k*2][j])
Внутренний цикл
Цикл по столбцам
матрицы B
Цикл по строкам
матрицы A
N % 2 != 0 - ?
Да
temp = N-1
Цикл по строкам
матрицы A,
i от 0 до M-1
Цикл по столбцам
матрицы B,
j от 0 до Q-1
В
Г

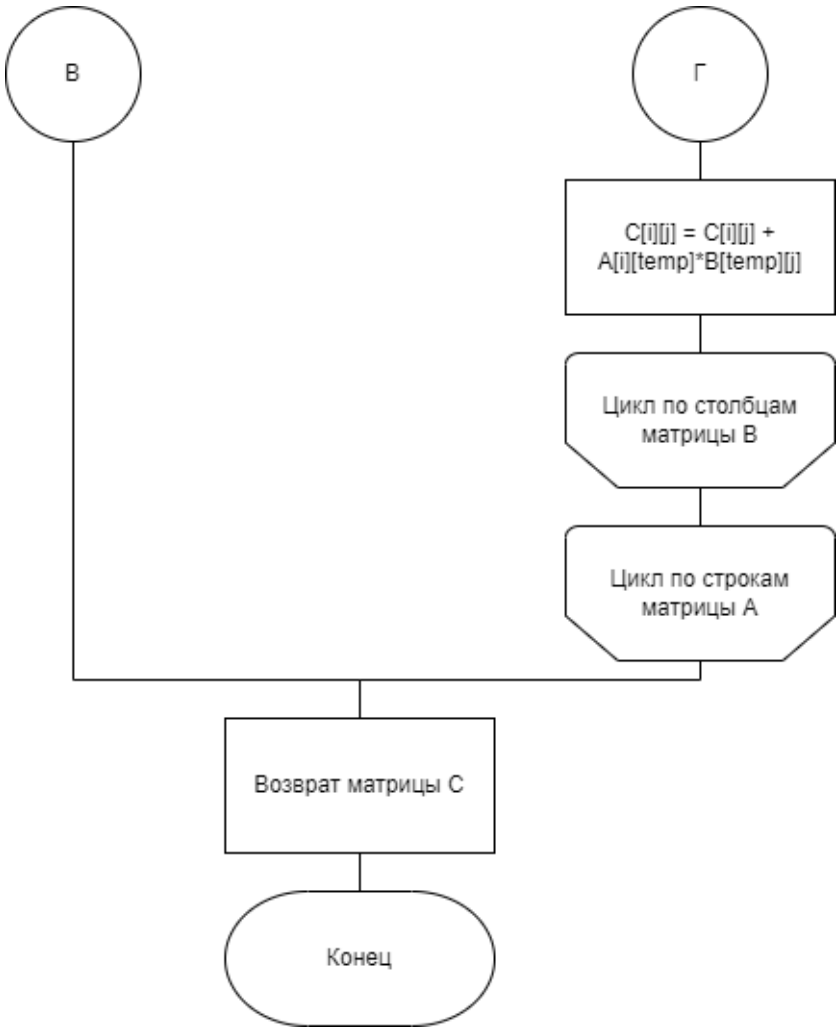
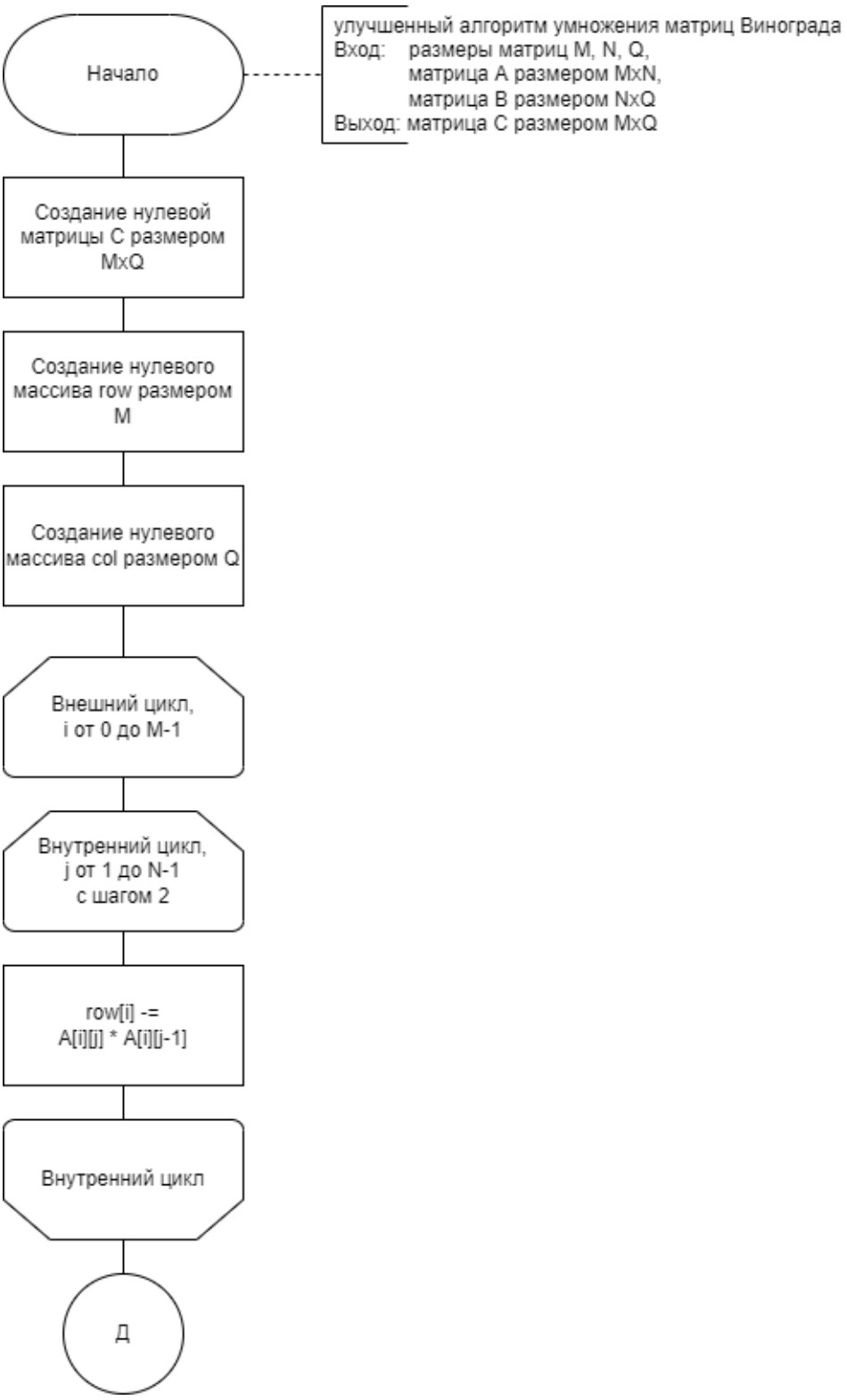
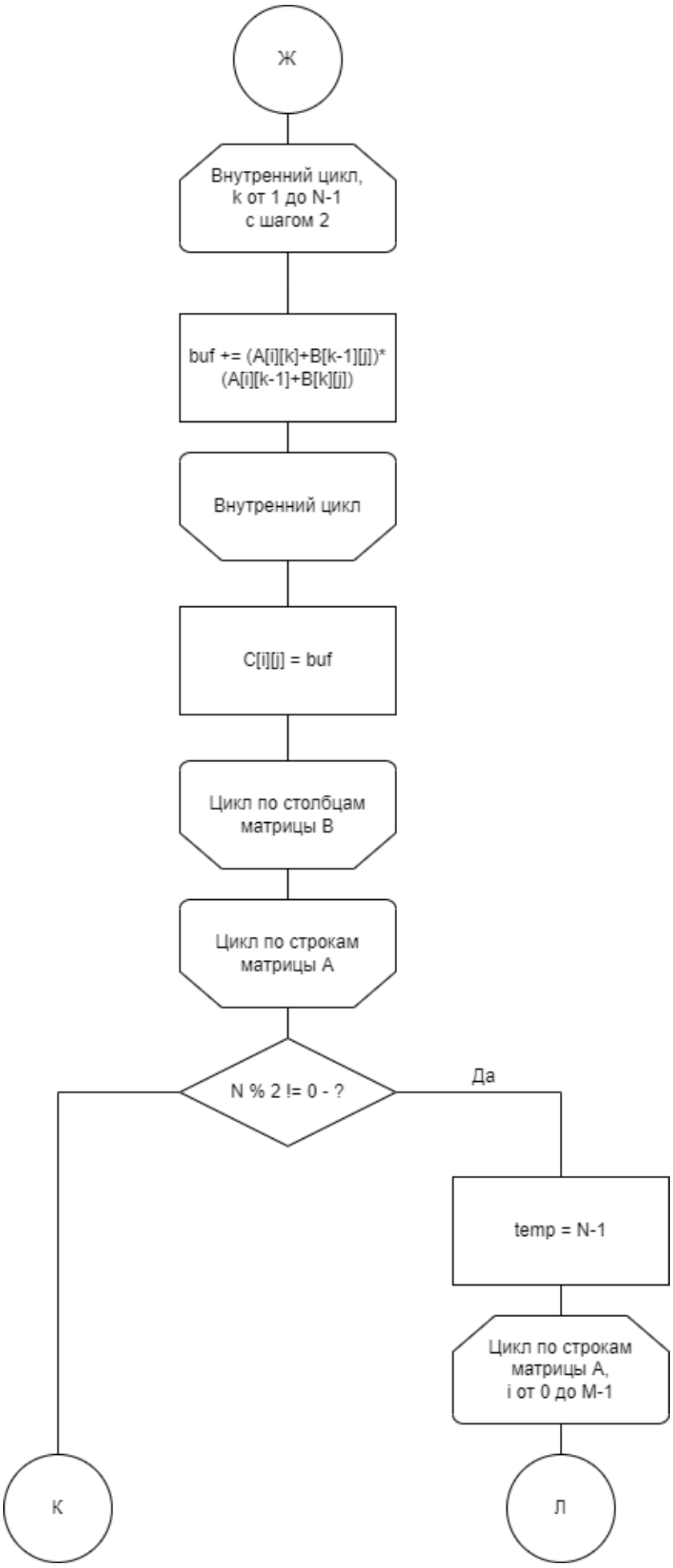


Рисунок 2.2 — Алгоритм умножения матриц Копперсмита–Винограда


Начало
улучшенный алгоритм умножения матриц Винограда
Вход: размеры матриц M, N, Q,
матрица A размером MxN,
матрица B размером NxQ
Выход: матрица C размером MxQ
Создание нулевой матрицы C размером MxQ
Создание нулевого массива row размером M
Создание нулевого массива col размером Q
Внешний цикл, i от 0 до M-1
Внутренний цикл, j от 1 до N-1 с шагом 2
row[i] -= A[i][j] * A[i][j-1]
Внутренний цикл
Д


Д
Внешний цикл
Внешний цикл,
j от 0 до Q-1
Внутренний цикл,
i от 1 до N-1
с шагом 2
col[j] -=
B[i][j] * B[i-1][j]
Внутренний цикл
Внешний цикл
Цикл по строкам
матрицы A,
i от 0 до M-1
Цикл по столбцам
матрицы B,
j от 0 до Q-1
buf = row[i] + col[j]
Ж


Ж
Внутренний цикл,
k от 1 до N-1
с шагом 2
buf += (A[i][k]+B[k-1][j])*
(A[i][k-1]+B[k][j])
Внутренний цикл
C[i][j] = buf
Цикл по столбцам
матрицы B
Цикл по строкам
матрицы A
N % 2 != 0 - ?
Да
temp = N-1
Цикл по строкам
матрицы A,
i от 0 до M-1
К
Л

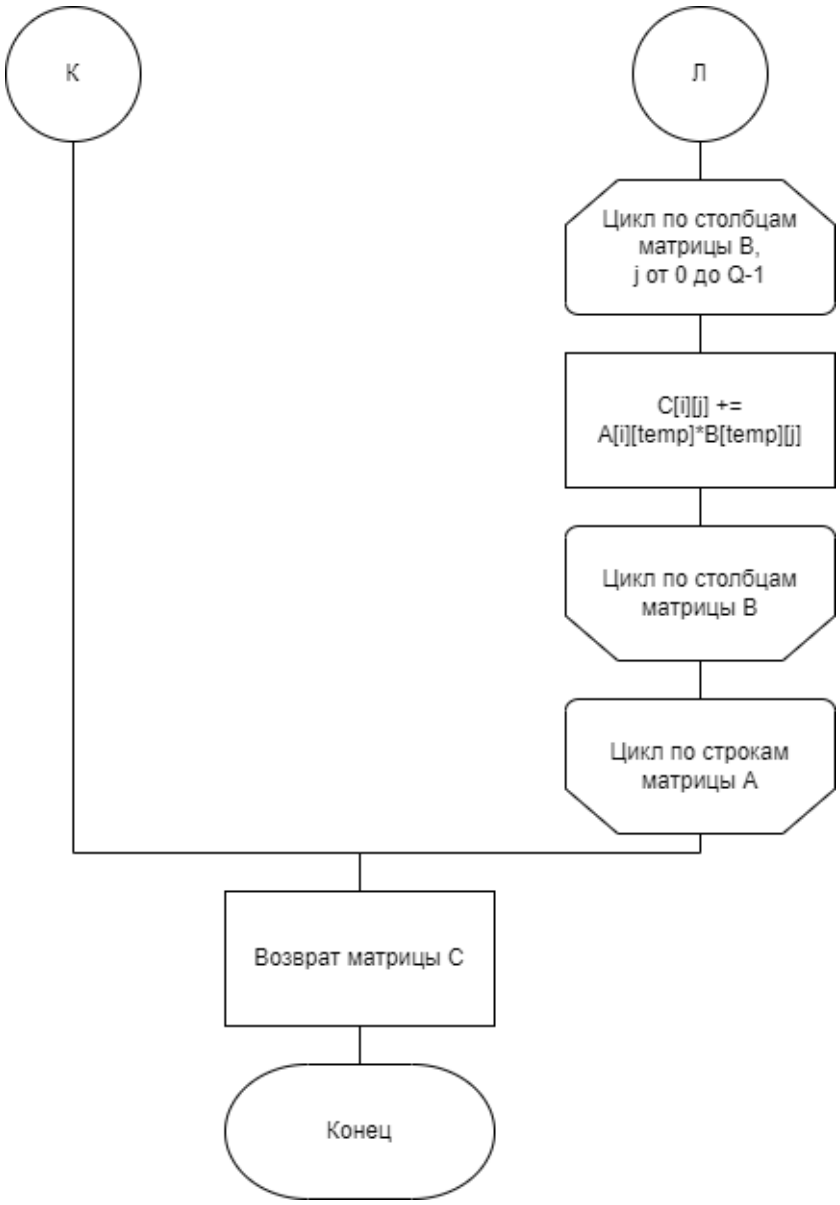


Рисунок 2.3 — Оптимизированный алгоритм умножения матриц Копперсмита–Винограда

2.2 Модель вычислений трудоемкости алгоритмов

Будем использовать следующую модель вычислений трудоемкости алгоритмов:

1. Трудоемкость базовых операций.

   Трудоемкость операций из списка (2.1) равна 2.

$$*, /, \%, * =, / = \tag{2.1}$$

   Операции из списка (2.2) имеют трудоемкость 1.

$$=, +, -, + =, - =, ==, ! =, <, >, <=, >=, ||, \&, [] \tag{2.2}$$

2. Трудоемкость условного оператора.

   Трудоемкость условного перехода равна 0, а оператора if условие then A else B рассчитывается, как (2.3) в лучшем случае и как (2.4) в худщем случае.

$$f_{\text{if}} = f_{\text{условия}} + min(f_{\text{A}}, f_{\text{B}}) \tag{2.3}$$

$$f_{\text{if}} = f_{\text{условия}} + max(f_{\text{A}}, f_{\text{B}}) \tag{2.4}$$

   где $f_{\text{A}}, f_{\text{B}}$ — трудоемкости блоков операций A и B соответственно.

3. Трудоемкость цикла.

   Она рассчитывается, как (2.5).

$$f_{\text{for}} = f_{\text{инициализации}} + f_{\text{условия}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.5}$$

4. Трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов умножения матриц

Пусть размер матрицы A во всех вычислениях обозначается как $M \times N$, а размер матрицы B — $N \times Q$. Трудоемкость создания результирующей матрицы будет опущена, поскольку эта операция одинакова для всех алгоритмов умножения матриц.

2.3.1 Стандартный алгоритм

Трудоёмкость стандартного алгоритма:

- трудоёмкость цикла $i \in [0..M)$:

$$f_i = 2 + M \cdot (2 + f_j) \tag{2.6}$$

- трудоёмкость цикла $j \in [0..Q)$:

$$f_j = 2 + Q \cdot (2 + f_k) \tag{2.7}$$

- трудоёмкость цикла $k \in [0..N)$:

$$f_k = 2 + N \cdot (2 + 12) \tag{2.8}$$

Трудоёмкость стандартного алгоритма умножения матриц (2.9):

$$\begin{aligned} f_{simple} &= 2 + M \cdot (2 + 2 + Q \cdot (2 + 2 + N \cdot (2 + 12))) = \\ &= 14MNQ + 4QM + 4M + 2 \approx 14QMN = O(QMN) \end{aligned} \tag{2.9}$$

2.3.2 Алгоритм Копперсмита–Винограда

Трудоёмкость алгоритма Копперсмита–Винограда:

- трудоемкость циклов инициализации векторов:

$$f_{initrow} = 2 + M \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 15)), \tag{2.10}$$

$$f_{initcol} = 2 + Q \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 15)), \tag{2.11}$$

- трудоемкость цикла инициализации результирующей матрицы:

$$f_{res} = 2 + M \cdot (2 + 2 + Q \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (2 + 28))), \tag{2.12}$$

- трудоемкость условного оператора:

$$f_{if} = 3 + \begin{bmatrix} 0, & \text{в лучшем случае} \\ 14MQ + 4M + 4, & \text{в худшем случае.} \end{bmatrix} \tag{2.13}$$

Трудоёмкость в **лучшем** случае (2.14).

$$\begin{gathered} f_{best} = 2 + M \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 15)) + \\ +2 + Q \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 15)) + \\ +2 + M \cdot (2 + 2 + Q \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (2 + 28))) + 3 = \\ = 15MNQ + 11MQ + 8.5MN + 8.5NQ + 8M + 4Q + 9 \approx \\ \approx 15MNQ = O(MNQ) \end{gathered} \tag{2.14}$$

Трудоёмкость в **худшем** случае (2.15).

$$\begin{gathered} f_{worst} = 15MNQ + 25MQ + 8.5MN + 8.5NQ + 12M + 4Q + 13 \approx \\ \approx 15MNQ = O(MNQ) \end{gathered} \tag{2.15}$$

2.3.3 Оптимизированный алгоритм Копперсмита–Винограда

Трудоёмкость оптимизированного алгоритма Копперсмита–Винограда:

- трудоемкость циклов инициализации векторов:

$$f_{initrow} = 2 + M \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 9)), \tag{2.16}$$

$$f_{initcol} = 2 + Q \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 9)), \tag{2.17}$$

- трудоемкость цикла инициализации результирующей матрицы:

$$f_{res} = 2 + M \cdot (2 + 2 + Q \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (2 + 15))), \tag{2.18}$$

- трудоемкость условного оператора:

$$f_{if} = 3 + \begin{bmatrix} 0, & \text{в лучшем случае} \\ 11MQ + 4M + 4, & \text{в худшем случае.} \end{bmatrix} \tag{2.19}$$

Трудоёмкость в **лучшем** случае (2.20).

$$\begin{gathered} f_{best} = 2 + M \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 9)) + \\ +2 + Q \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 9)) + \\ +2 + M \cdot (2 + 2 + Q \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (2 + 15))) + 3 = \\ = 8.5MNQ + 11MQ + 5.5MN + 5.5NQ + 8M + 4Q + 9 \approx \\ \approx 8.5MNQ = O(MNQ) \end{gathered} \tag{2.20}$$

Трудоёмкость в **худшем** случае (2.21).

$$\begin{gathered} f_{worst} = 8.5MNQ + 22MQ + 5.5MN + 5.5NQ + 12M + 4Q + 13 \approx \\ \approx 8.5MNQ = O(MNQ) \end{gathered} \tag{2.21}$$

2.4 Вывод по конструкторской части

На основе теоретических данных, полученных из аналитического раздела, были построены схемы алгоритмов умножения матриц. Оценены их трудоемкости.

При умножении матриц трудоемкость будет зависеть от размеров матриц в каждом конкретном случае. Из расчетов видно, что оптимизация алгоритма Копперсмита–Винограда позволила достичь меньшей трудоемкости в сравнении со стандартным алгоритмом.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход ПО получает размеры двух матриц, а также сами элементы матриц;
- на выходе — результирующая матрица, полученная путем умножения двух исходных.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования `Python` [1]. Это обусловлено знанием возможностей языка, что обеспечит высокую скорость написания программы без потери ее качества.

В качестве среды разработки выбрана `Visual Studio Code` [3]. Удобства написания кода и его автодополнения стали ключевыми при выборе.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведены реализации трёх алгоритмов умножения матриц.

Листинг 3.1 — Алгоритм Копперсмита–Винограда

```
def dotMatrixVinograd(a, b):
  if (len(b) != len(a[0])):
    raise ValueError

  m = len(a)
  n = len(a[0])
  q = len(b[0])
  c = [[0] * q for i in range(m)]

  row = [0] * m
  col = [0] * q

  temp = n // 2
  for i in range(m):
    for j in range(temp):
      row[i] = row[i] + a[i][2*j] * a[i][2*j + 1]

  for j in range(q):
    for i in range(temp):
      col[j] = col[j] + b[2*i][j] * b[2*i + 1][j]

  for i in range(m):
    for j in range(q):
      c[i][j] = -row[i] - col[j]
      for k in range(temp):
        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 n % 2:
    temp = n - 1
    for i in range(m):
      for j in range(q):
        c[i][j] = c[i][j] + a[i][temp] * b[temp][j]

  return c
```

Листинг 3.2 — Оптимизированный алгоритм Копперсмита–Винограда

```
def dotMatrixVinogradOpt(a, b):
  if (len(b) != len(a[0])):
    raise ValueError

  m = len(a)
  n = len(a[0])
  q = len(b[0])
  c = [[0] * q for i in range(m)]

  row = [0] * m
  col = [0] * q

  for i in range(m):
    for j in range(1, n, 2):
      row[i] -= a[i][j] * a[i][j - 1]

  for j in range(q):
    for i in range(1, n, 2):
      col[j] -= b[i][j] * b[i - 1][j]

  for i in range(m):
    for j in range(q):
      buf = row[i] + col[j]
      for k in range(1, n, 2):
        buf += (a[i][k - 1] + b[k][j]) * (a[i][k] + b[k
            -1][j])
      c[i][j] = buf

  if n % 2:
    temp = n - 1
    for i in range(m):
      for j in range(q):
        c[i][j] += a[i][temp] * b[temp][j]

  return c
```

Листинг 3.3 — Стандартный алгоритм умножения матриц

```
def dotMatrix(a, b):
  if (len(b) != len(a[0])):
    raise ValueError

  m = len(a)
  n = len(a[0])
  q = len(b[0])
  c = [[0] * q for i 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
```

3.4 Тестовые данные

В таблице (3.1) приведены тесты для функций, реализующих алгоритмы умножения матриц.

Таблица 3.1 — Тестирование функций

Входные матрицы	Результат	Ожидаемый результат
$\begin{pmatrix} 2 & 1 \\ 3 & 3 \\ -1 & 2 \end{pmatrix} \times \begin{pmatrix} -5 & 1 \\ 3 & -2 \end{pmatrix}$	$\begin{pmatrix} -7 & 0 \\ -6 & -3 \\ 11 & -5 \end{pmatrix}$	$\begin{pmatrix} -7 & 0 \\ -6 & -3 \\ 11 & -5 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} -5 & 1 & 3 \end{pmatrix} \times \begin{pmatrix} 2 \\ -3 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 5 \end{pmatrix}$
$\begin{pmatrix} 2 \\ -3 \\ 6 \end{pmatrix} \times \begin{pmatrix} -5 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} -10 & 2 & 6 \\ 15 & -3 & -9 \\ -30 & 6 & 18 \end{pmatrix}$	$\begin{pmatrix} -10 & 2 & 6 \\ 15 & -3 & -9 \\ -30 & 6 & 18 \end{pmatrix}$
$\begin{pmatrix} 6 \end{pmatrix} \times \begin{pmatrix} -5 \end{pmatrix}$	$\begin{pmatrix} -30 \end{pmatrix}$	$\begin{pmatrix} -30 \end{pmatrix}$

Все тесты пройдены успешно.

3.5 Вывод по технологической части

В данном разделе были реализованы и протестированы алгоритмы умножения матриц: стандартный алгоритм, алгоритм Копперсмита–Винограда и оптимизированный алгоритм Копперсмита–Винограда.

4 Исследовательская часть

Исследуем быстродействие алгоритмов умножения матриц на практических тестах.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система Windows 10 64-разрядная.
- Оперативная память 16 ГБ.
- Процессор Intel(R) Core(TM) i5-4690 @ 3.50ГГц.

4.2 Время выполнения реализаций алгоритмов

Время выполнения реализаций алгоритмов замерялось с помощью специальной функции `process_time()` [2] из модуля `time`, которая возвращает значение в долях секунды процессорного времени текущего процесса. Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов.

На рисунках 4.1 - 4.4 показаны результаты работы ПО. Все значения отображены в секундах.

Лучший случай

Размер	Стандартный	Винограда	Винограда оптим.
100	0.17188	0.21875	0.14062
200	1.48438	1.78125	1.14062
300	5.40625	6.25000	3.92188
400	13.01562	15.40625	10.15625
500	27.01562	32.98438	21.78125
600	48.56250	58.82812	39.62500
700	79.14062	97.00000	65.37500
800	118.65625	146.92188	99.62500
900	173.54688	214.35938	144.07812
1000	241.37500	293.07812	201.20312

Рисунок 4.1 — Замеры на четных размерах матриц

Худший случай

Размер	Стандартный	Винограда	Винограда оптим.
101	0.21875	0.21875	0.14062
201	1.70312	1.75000	1.14062
301	5.34375	6.09375	3.98438
401	13.62500	15.62500	10.20312
501	27.57812	32.82812	21.79688
601	48.98438	59.09375	40.10938
701	79.50000	96.23438	65.40625
801	120.64062	147.85938	99.73438
901	173.56250	210.25000	144.75000
1001	240.62500	294.51562	200.78125

Рисунок 4.2 — Замеры на нечетных размерах матриц

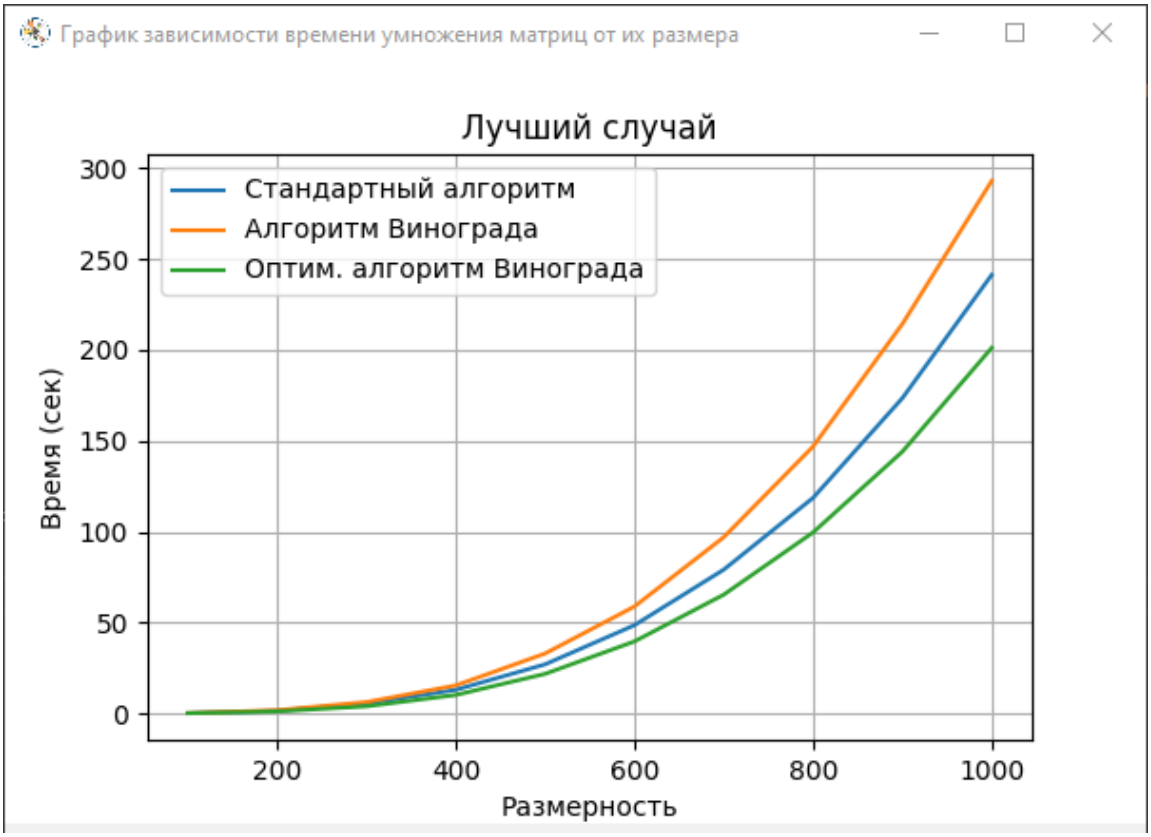


Рисунок 4.3 — Время выполнения реализаций алгоритмов на четных размерах матриц

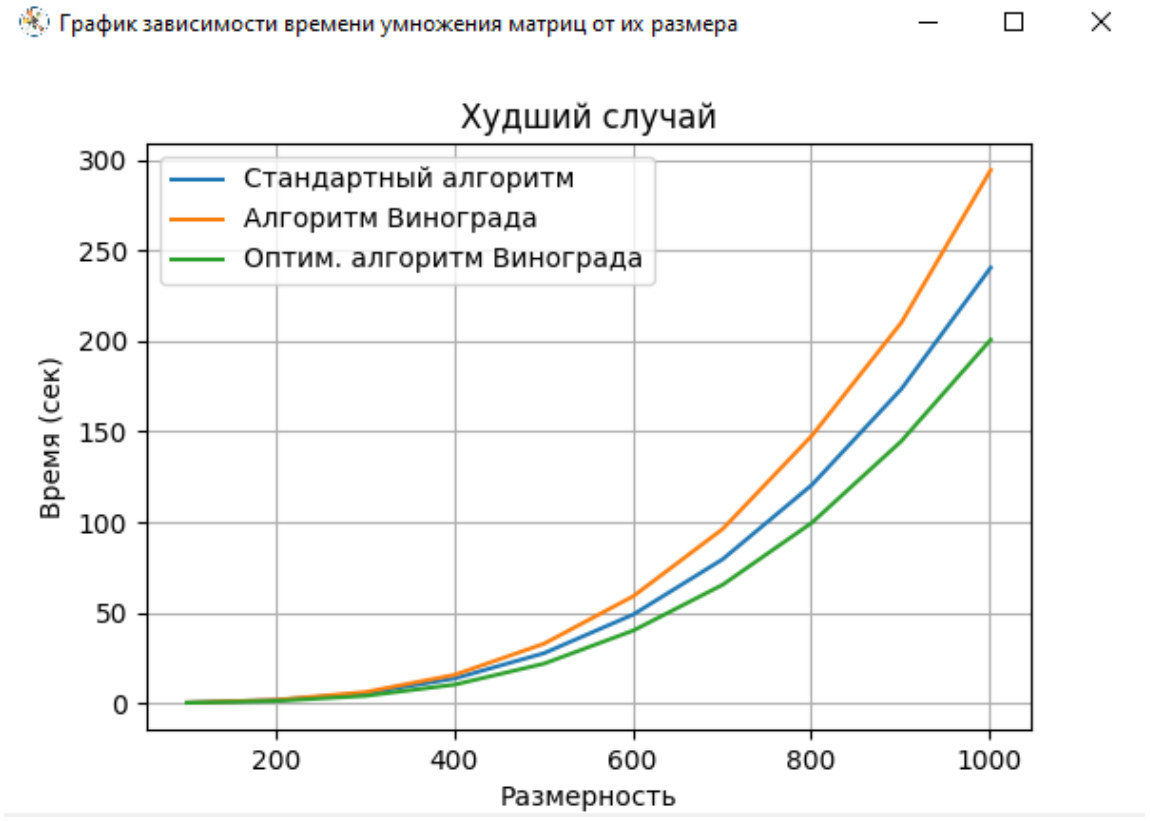


Рисунок 4.4 — Время выполнения реализаций алгоритмов на нечетных размерах матриц

4.3 Вывод по исследовательской части

По результатам замеров самым быстрым алгоритмом оказался оптимизированный алгоритм Копперсмита–Винограда. При этом исходная версия алгоритма Копперсмита–Винограда уступает стандартному алгоритму умножения матриц как на четных, так и на нечетных размерностях матриц.

Заключение

В ходе проделанной работы была достигнута поставленная цель и решены следующие задачи:

- были изучены алгоритмы умножения матриц: стандартный алгоритм, алгоритм Копперсмита–Винограда, оптимизированный алгоритм Копперсмита–Винограда;
- был проведён сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- эти алгоритмы были реализованы и успешно протестированы;
- был проведён сравнительный анализ алгоритмов на основе экспериментальных данных.

На основании анализа трудоемкости алгоритмов в выбранной модели вычислений было показано, что оптимизированный алгоритм Копперсмита–Винограда имеет меньшую трудоемкость. В среднем трудоемкость всех рассматриваемых алгоритмов имеет кубический характер зависимости от линейного размера матрицы. С помощью замеров времени выполнения реализаций алгоритмов умножения матриц была доказана посчитанная теоретически трудоемкость алгоримов. Таким образом, оптимизированный алгоритм Копперсмита–Винограда работает быстрее остальных рассматриваемых алгоритмов умножения матриц, при этом его исходная версия работает медленнее стандартного алгоритма.

Литература

[1] Python [Электронный ресурс]. Режим доступа: https://python.org. Дата обращения: 22.10.2021.

[2] Модуль time [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html. Дата обращения: 22.10.2021.

[3] Visual Studio Code - Code Editing [Электронный ресурс]. Режим доступа: https://code.visualstudio.com. Дата обращения: 22.10.2021.

[4] Coppersmith D., Winograd S. 1990, Matrix multiplication via arithmetic Progressions, J. Symbolic Computation., Т. 9, Изд. 3, С. 251-280.