**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»

# Лабораторная работа № 3
# по дисциплине «Анализ алгоритмов»

**Тема** Поиск в массиве

**Студент** Бугаков И. С.

**Группа** ИУ7-54Б

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

Москва, 2024

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

В данной лабораторной работе будут рассмотрены алгоритмы умножения матриц. Операции с матрицами используются при компьютерном решении задач аналитической геометрии, в компьютерной графике [7], машинном обучении [3, 6].

Целью данной лабораторной работы является анализ алгоритмов умножения матриц: стандартного алгоритма и алгоритма Винограда с оптимизациями и без.

Задачи лабораторной работы:

1) описание стандартного алгоритма и Винограда;
2) анализ трудоемкости указанных алгоритмов;
3) реализация указанных алгоритмов;
4) тестирование реализаций;
5) провести замеры и последующий анализ временных затрат реализаций.

# 1 Аналитическая часть

Долгое время считалось, что перемножение матриц быстрее, чем по определению за $O(n^3)$ невозможно. Однако, Штрассен показал [4], что вычисление может быть выполнено за $O(n^{2,81})$. Позднее Копперсмит и Виноград [5] улучшили эту оценку до $O(n^{2,3755})$.

## 1.1 Стандартный алгоритм умножения

### 1.1.1 Описание алгоритма

Данный алгоритм напрямую соответствует определению операции умножения матриц. Пусть матрицы $A$ и $B$ представлены в виде 1.1. Тогда матрица $C = AB$ представлена в формуле 1.2, а ее элементы определяются формулой 1.3.

$$A_{n,m} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}, B_{m,r} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,r} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,r} \end{pmatrix} \tag{1.1}$$

$$C_{n,r} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,r} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,r} \end{pmatrix} \tag{1.2}$$

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot bk, j, 1 \leq i \leq n, 1 \leq j \leq r \tag{1.3}$$

### 1.1.2 Асимптотика

Асимптотика такого алгоритма составляет $O(n^3)$, т. к. необходимо $O(n)$ операций для вычисления каждого из $n^2$ элементов результирующей матрицы.

## 1.2 Алгоритм Винограда

Алгоритм Копперсмита — Винограда стал развитием идеи сокращения числа операций умножения. Идея алгоритма заключается в том, что элемент результирующей матрицы $c_{i,j}$

представлен скалярным произведением 1.4.

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot b_{k,j} = \begin{pmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,m} \end{pmatrix} \times \begin{pmatrix} b_{1,j} \\ b_{2,j} \\ \vdots \\ b_{m,j} \end{pmatrix} \tag{1.4}$$

Данное скалярное произведение представимо в виде 1.5.

$$c_{i,j} = \sum_{k=1}^{\frac{r}{2}} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{\frac{r}{2}} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{\frac{r}{2}} b_{2k-1,j}b_{2k,j} \tag{1.5}$$

В формуле 1.5 второе и третье слагаемое могут быть рассчитаны заранее для каждой строки и каждого столбца матриц $A$ и $B$ соответственно. В дальнейшем будут использованы уже рассчитанные значения, что уменьшит трудоемкость.

При нечетном размере матриц, необходимо также добавить произведения крайних элементов соответствующих строк и столбцов для корректности вычисления.

### 1.2.1 Асимптотика

Асимптотика представленного алгоритма составляет $O(n^{2,3755})$ [5].

## 1.3 Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут быть использованы следующие подходы:

1) операции умножения и деления на степени двойки могут быть заменены побитовыми операциями;

2) использование операций $+=$ и $-=$, чтобы избежать лишних обращений к памяти;

3) внести вычисление произведений крайних элементов в основные вычисления алгоритма;

4) вынести первые итерации из циклов как инициализации соответствующих переменных.

### 1.3.1 Асимптотика

Асимптотика, как и в случае не оптимизированного алгоритма составляет $O(n^{2,3755})$, однако, константа, которую не учитывает $O$-нотация, будет меньше.

# Вывод

В данной части работы были описаны стандартный алгоритм перемножения матриц и алгоритм Винограда, приведены асимптотики.

# 2 Конструкторская часть

## 2.1 Схемы алгоритмов

Ниже приведены схемы алгоритмов, упомянутых в аналитической части:

— на рисунке 2.1 — схема стандартного алгоритма умножения матриц;

— на рисунке 2.2 — схема алгоритма Винограда;

— на рисунке 2.3 — схема алгоритма Винограда с оптимизациями.

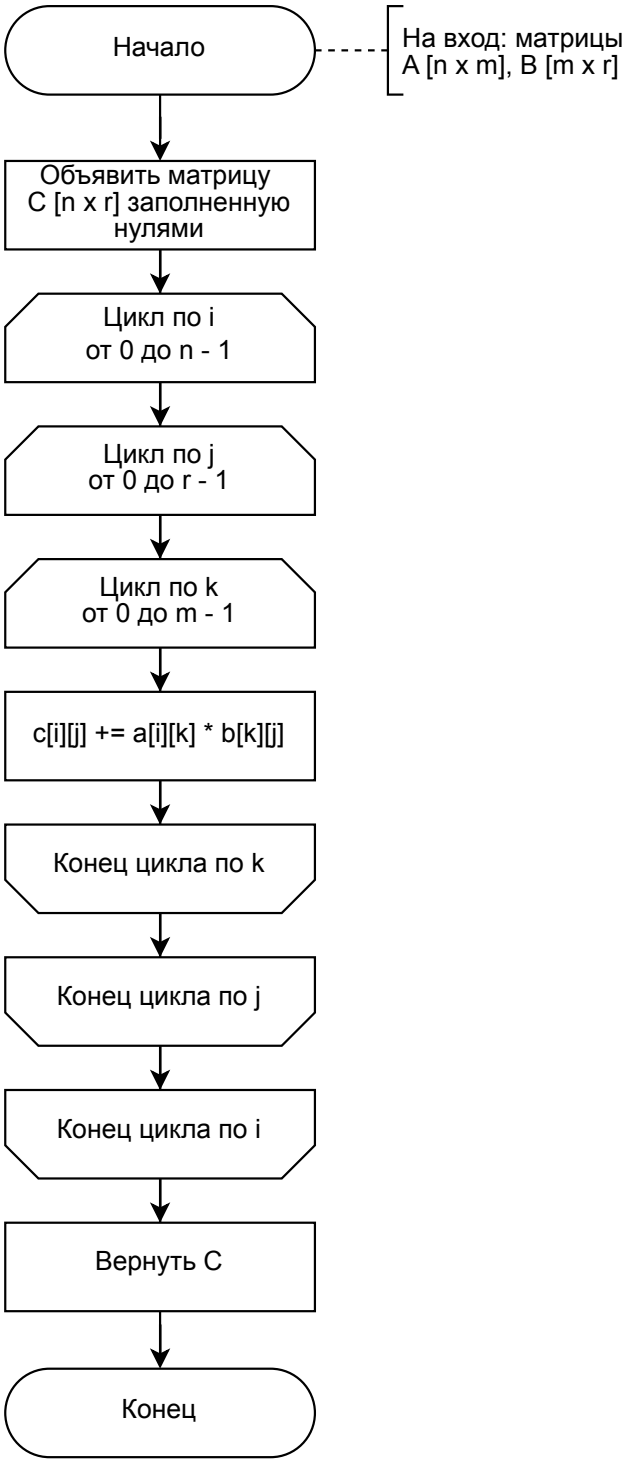


Рисунок 2.1 — Схема алгоритма линейного поиска

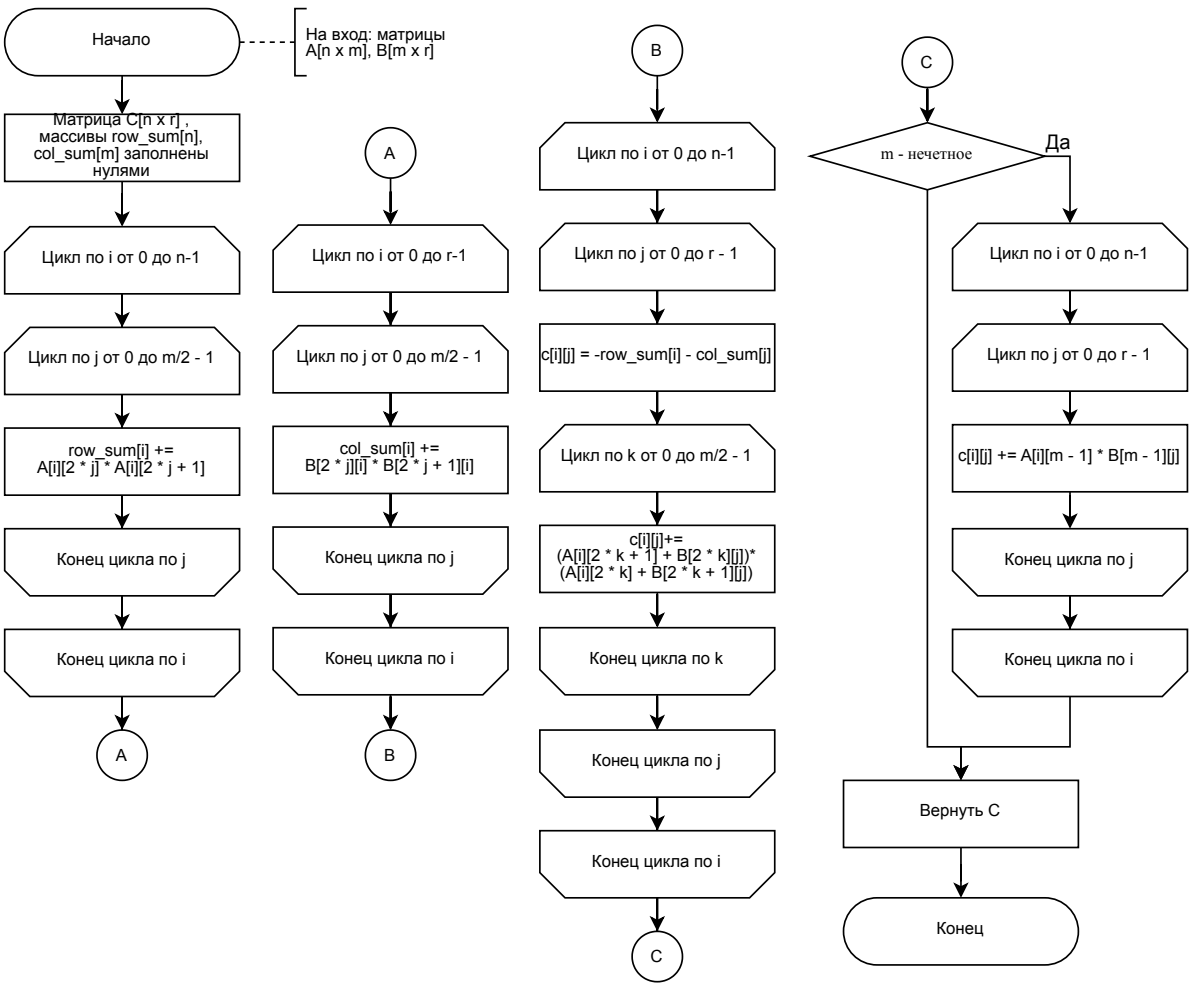


Рисунок 2.2 — Схема алгоритма Винограда

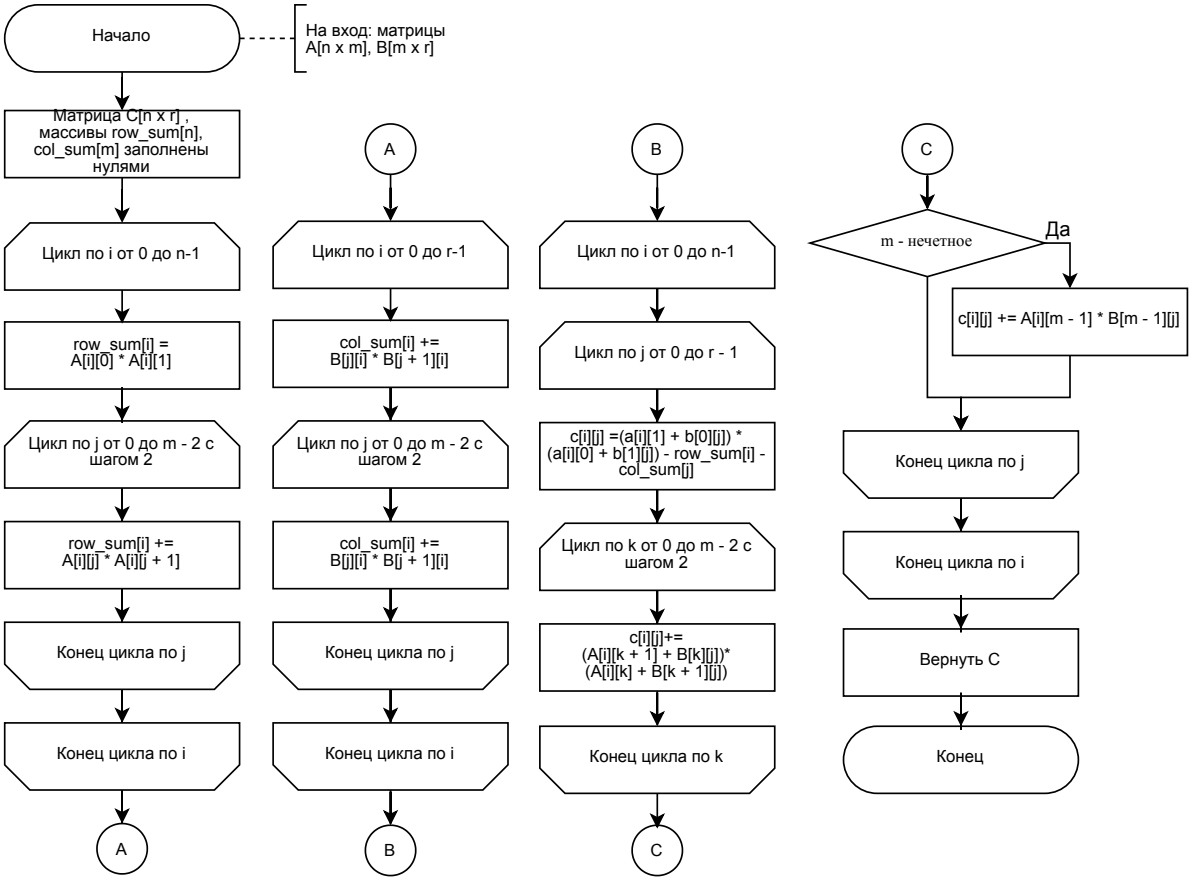


Рисунок 2.3 — Схема алгоритма Винограда с оптимизациями

## 2.2 Требования к реализации алгоритмов

Реализации алгоритмов должны удовлетворять следующим общим требованиям:

— входные данные: два двумерных массива целых чисел, число колонок в первом должно совпадать с числом строк во втором;

— выходные данные: матрица являющаяся результатом умножения исходных матриц представленных двумерными массивами.

## 2.3 Оценка трудоемкости

### 2.3.1 Модель вычислений

Модель вычислений принимается следующей:

— для указанных ниже операций, стоимость принимается равной 1 (— подразумевается

как унарный и бинарный, $*$ подразумевается умножение или разыменованием указателя):

$$+,\ -,\ *,\ /,\ ++,\ --,\ +=,\ -=,\ <,\ >,\ >=,\ <=,\ =,\ ==,\ [\,],\ !=$$

— трудоемкость цикла от 0 до n-1 включительно:

$$f_{cycle} = f_{initialize} + f_{condition} + n \cdot (f_{update} + f_{condition} + f_{body})$$

— трудоемкость условного оператора:

$$\begin{aligned} f_{worst} &= f_{condtition} + max(f_1, f_2) \\ f_{best} &= f_{condition} + min(f_1, f_2) \end{aligned}$$

### 2.3.2 Стандартный алгоритм умножения матриц

Трудоемкость этого алгоритма не имеет лучшего или худшего случая, так как все три цикла будут всегда выполнены полностью. Соответственно трудоемкость приведена в формуле 2.1.

$$\begin{aligned} f_{standard} = 1 + 1 + n \cdot (1 + 1 + (1 + 1 + r \cdot (1 + 1 + (1 + 1 + m \cdot (1 + 1 + 8))))) = \\ = 10nrm + 4nr + 4n + 2 \end{aligned} \tag{2.1}$$

### 2.3.3 Алгоритм Винограда

Трудоемкость алгоритма Винограда имеет два случая в зависимости от четности числа колонок первой матрицы.

Трудоемкости отдельных частей алгоритма:

— вычисление суммы по строкам первой матрицы:

$$f_{str\ sum} = 1 + 1 + n \cdot (1 + 1 + \frac{m}{2} \cdot 12) = 2 + 2n + 6nm \tag{2.2}$$

— вычисление суммы по столбцам второй матрицы:

$$f_{col\ sum} = 1 + 1 + r \cdot (1 + 1 + \frac{m}{2} \cdot 12) = 2 + 2r + 6rm \tag{2.3}$$

— вычисление всего произведения:

$$\begin{aligned} f_{mul} = 1 + 1 + n \cdot (1 + 1 + r \cdot (7 + 1 + 1 + \tfrac{m}{2} \cdot 23)) = \\ = 2 + 2n + 9rn + \tfrac{23}{2}nrm \end{aligned} \tag{2.4}$$

— вычисление дополнительных произведений крайних элементов имеет два случая:

худший, если $m$ нечетное, и лучший, если $m$ четное:

$$\begin{gathered} f_{edge\ best} = 0 \\ f_{edge\ worst} = 1 + 1 + n \cdot (1 + 1 + r \cdot 12) = 2 + 2n + 13rn \end{gathered} \tag{2.5}$$

Сложением выражений 2.2, 2.3, 2.4, 2.5 получены формулы лучшего и худшего случая алгоритма Винограда:

$$f_{best} = 6 + 4n + 2r + 6nm + 6rn + 6rm + \frac{23}{2}nrm \tag{2.6}$$

$$f_{worst} = 8 + 6n + 2r + 6nm + 6rm + 19rn + \frac{23}{2}nrm \tag{2.7}$$

В среднем случае вероятности того, что $m$ четно или нечетно равны и составляют $p = 0,5$. В таком случае средняя трудоемкость:

$$f_{mean} = 7 + 5n + 6nm + 6rm + \frac{25}{2}rn + \frac{23}{2}nrm \tag{2.8}$$

### 2.3.4 Алгоритм Винограда с оптимизацией

Аналогично алгоритму без оптимизаций расчет трудоемкости может быть произведен по частям:

— вычисление суммы по строкам первой матрицы:

$$f_{str\ sum} = 1 + 1 + n \cdot (7 + 1 + 1 + (\frac{m}{2} - 1) \cdot 12) = 2 - 3n + 6nm \tag{2.9}$$

— вычисление суммы по столбцам второй матрицы:

$$f_{col\ sum} = 1 + 1 + r \cdot (7 + 1 + 1 + (\frac{m}{2} - 1) \cdot 12) = 2 - 3r + 6rm \tag{2.10}$$

— вычисление всего произведения имеет два случая в зависимости от четности $m$:

$$\begin{gathered} f_{best} = 1 + 1 + n(1 + 1 + r(18 + 1 + 1 + (\frac{m}{2} - 1) \cdot 16 + 0)) = \\ = 2 + 2n + 4rn + 8nrm \\ f_{worst} = 1 + 1 + n(1 + 1 + r(18 + 1 + 1 + (\frac{m}{2} - 1) \cdot 16 + 10)) = \\ = 2 + 2n + 14rn + 8nrm \end{gathered} \tag{2.11}$$

Общая трудоемкость оптимизированного алгоритма получается сложением полученных ранее выражений 2.9, 2.10, 2.11:

$$\begin{gathered} f_{best} = 6 - n - 3r + 6nm + 6rm + 4rn + 8nrm \\ f_{worst} = 6 - n - 3r + 6nm + 6rm + 14rn + 8nrm \end{gathered} \tag{2.12}$$

В среднем трудоемкость получается следующей:

$$f_{mean} = 6 - n - 3r + 6nm + 6rm + 9rn + 8nrm \tag{2.13}$$

# Вывод

На основе аналитической части построены схемы алгоритмов, определены требования к их реализации, вычислены их трудоемкости.

# 3 Технологическая часть

## 3.1 Выбор языка программирования

Для реализации указанных алгоритмов был выбран язык C++, т. к. данный язык предоставляет возможность работать с динамической памятью с помощью контейнерных классов.

Для визуализации данных использовался язык Python, т. к. данный язык обладает широким выбором библиотек для выполнения этой задачи. В лабораторной работе использовалась библиотека matplotlib [2].

В качестве IDE был выбран Clion, т. к. обладает функциями подсветки синтаксиса и автодополнения для обоих указанных языков. Также в данной IDE реализована поддержка средства автоматизированной сборки проектов cmake.

## 3.2 Реализации алгоритмов

В листингах ниже представлены реализации указанных алгоритмов:

— в листинге 3.1 — реализация стандартного алгоритма умножения матриц;

— в листинге 3.2 — реализация алгоритма Винограда.

— в листинге 3.3 — оптимизированная реализация алгоритма Винограда

Листинг 3.1 — Стандартный алгоритм

```
vector<vector<int>> std_mtrx_mul(const vector<vector<int>> &a, const
    vector<vector<int>> &b) {
  vector<vector<int>> c(a.size(), vector<int>(b[0].size()));
  for (int i = 0; i < a.size(); ++i)
  for (int j = 0; j < b[0].size(); ++j)
  for (int k = 0; k < b.size(); ++k)
  c[i][j] += a[i][k] * b[k][j];
  return c;
}
```

Листинг 3.2 — Алгоритм Винограда

```
vector<vector<int>> winograd(const vector<vector<int>> &a, const
    vector<vector<int>> &b) {
  int n = a.size(), m = b.size(), r = b[0].size();
  vector<int> row_sum(n);
  vector<int> col_sum(r);
  vector<vector<int>> c(n, vector<int>(r));
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < m / 2; ++j)
  row_sum[i] = row_sum[i] + a[i][2 * j] * a[i][2 * j + 1];
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < m / 2; ++j)
  col_sum[i] = col_sum[i] + b[2 * j][i] * b[2 * j + 1][i];
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < r; ++j) {
    c[i][j] = -row_sum[i] - col_sum[j];
    for (int 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)
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < r; ++j)
  c[i][j] = c[i][j] + a[i][m - 1] * b[m - 1][j];
  return c;
}
```

Листинг 3.3 — Оптимизированная реализация алгоритма Винограда

```
vector<vector<int>> winograd_optimized(const vector<vector<int>> &a,
    const vector<vector<int>> &b) {
  int n = a.size(), m = b.size(), r = b[0].size();
  vector<int> row_sum(n);
  vector<int> col_sum(r);
  vector<vector<int>> c(n, vector<int>(r));
  for (int i = 0; i < n; ++i) {
    row_sum[i] = a[i][0] * a[i][1];
    for (int j = 2; j < m - 1; j += 2) {
      row_sum[i] += a[i][j] * a[i][j + 1];
    }
  }
  for (int i = 0; i < r; ++i) {
    col_sum[i] = b[0][i] * b[1][i];
    for (int j = 2; j < m - 1; j += 2) {
      col_sum[i] += b[j][i] * b[j + 1][i];
    }
  }
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < r; ++j) {
    c[i][j] = (a[i][1] + b[0][j]) * (a[i][0] + b[1][j]) - row_sum[i]
       - col_sum[j];
    for (int k = 2; k < m - 1; k += 2)
    c[i][j] += (a[i][k + 1] + b[k][j]) * (a[i][k] + b[k + 1][j]);
    if (m & 1)
    c[i][j] += a[i][m - 1] * b[m - 1][j];
  }
  return c;
}
```

## 3.3 Тестирование

Для тестирования алгоритмов были выбраны следующие тестовые случаи, представленные в таблице 3.1

Таблица 3.1 — Тестовые случаи для алгоритмов умножения матриц

| Входные данные | Ожидаемые выходные данные | Стандартный алгоритм | Алгоритм Винограда | Оптимизированная реализация алгоритма Винограда |
|---|---|---|---|---|
| $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$ $\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$ | $\begin{pmatrix}10 & 13\\22 & 29\end{pmatrix}$ | $\begin{pmatrix}10 & 13\\22 & 29\end{pmatrix}$ | $\begin{pmatrix}10 & 13\\22 & 29\end{pmatrix}$ | $\begin{pmatrix}10 & 13\\22 & 29\end{pmatrix}$ |
| $\begin{pmatrix}1 & 2\\3 & 3\\2 & 1\end{pmatrix}$ $\begin{pmatrix}1 & 3 & 2\\4 & 2 & 2\end{pmatrix}$ | $\begin{pmatrix}9 & 7 & 6\\15 & 15 & 12\\6 & 8 & 6\end{pmatrix}$ | $\begin{pmatrix}9 & 7 & 6\\15 & 15 & 12\\6 & 8 & 6\end{pmatrix}$ | $\begin{pmatrix}9 & 7 & 6\\15 & 15 & 12\\6 & 8 & 6\end{pmatrix}$ | $\begin{pmatrix}9 & 7 & 6\\15 & 15 & 12\\6 & 8 & 6\end{pmatrix}$ |

Для проведения тестирования использовалась библиотека Google Tests [1].

# Вывод

Были реализованы и протестированы стандартный алгоритм умножения матриц, алгоритм Винограда и его оптимизированная версия. Все тесты были успешно пройдены.

# 4 Исследовательская часть

## 4.1 Характеристики устройства

Исследования проводились на машине со следующими характеристиками:

— процессор Intel(R) Core(TM) i5-10210U, тактовая частота 1.60 ГГц;

— оперативная память: 16 ГБ;

— операционная система: Ubuntu 22.04.4 LTS.

## 4.2 Замеры процессорного времени выполнения алгоритмов

Замеры производились для квадратных матриц с линейными размерами от 0 до 500 для лучшего случая и от 25 до 475 с шагом 50 по 10 раз на каждую точку. На рисунке 4.1 приведены графики полученных зависимостей времени 10 запусков от линейного размера квадратной матрицы для лучшего случая, на рисунке 4.2 для худшего соответственно.

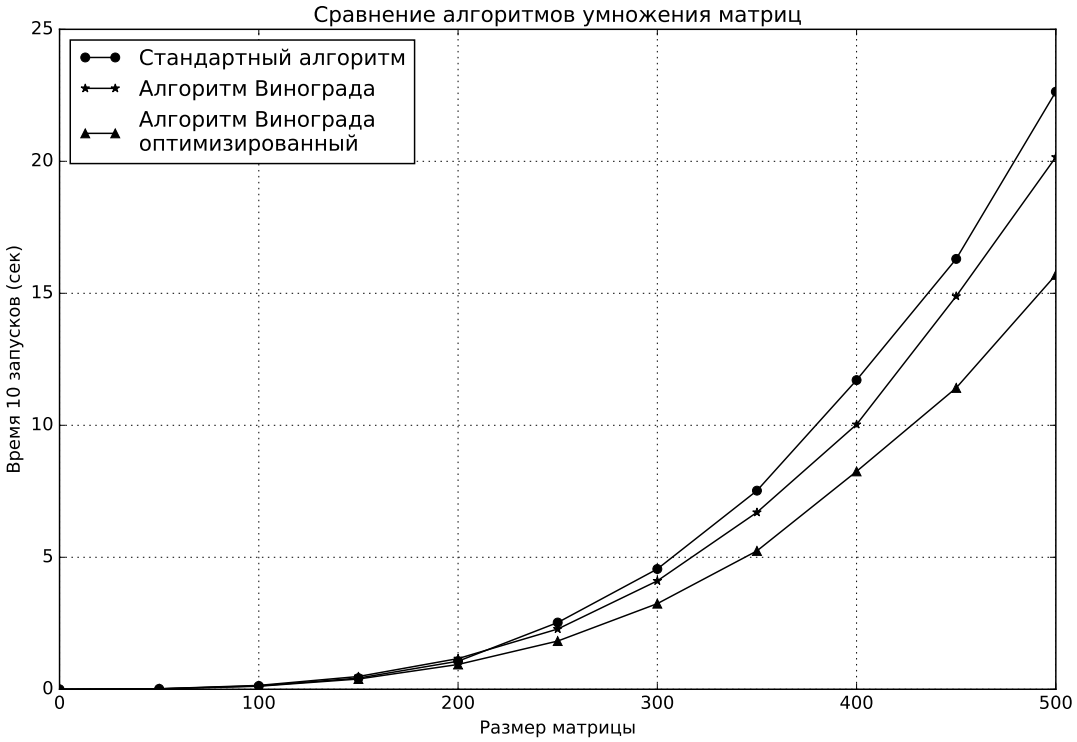


Рисунок 4.1 — График зависимости времени от линейного размера квадратных матриц в лучшем случае

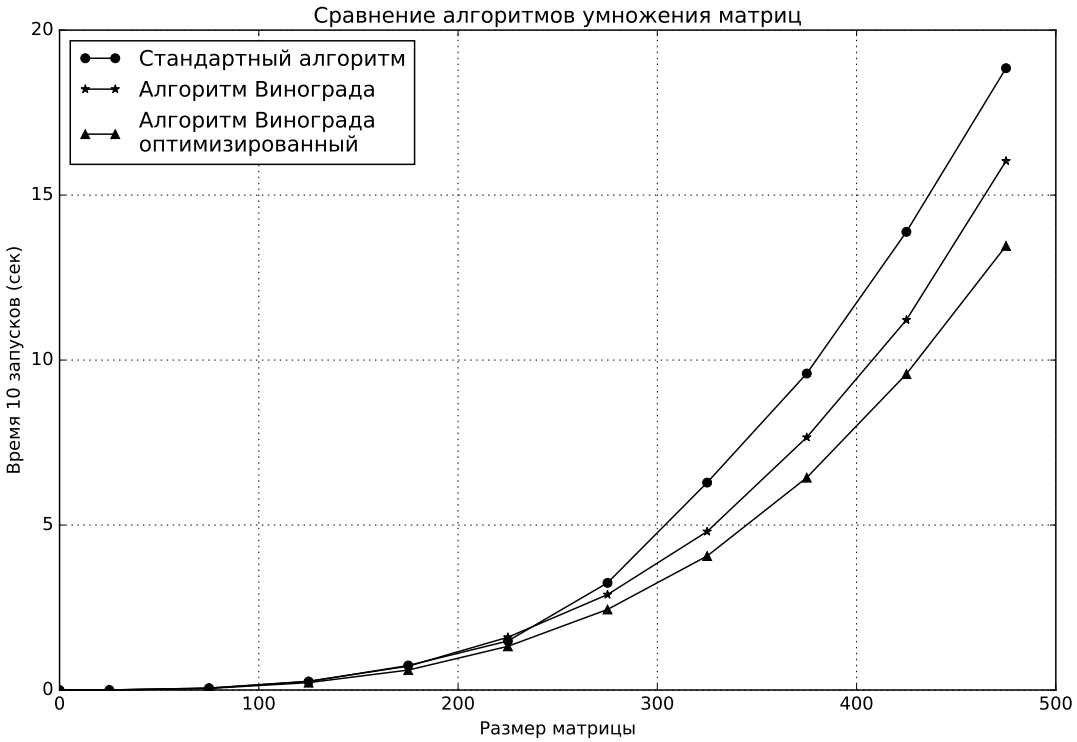


Рисунок 4.2 — График зависимости времени от линейного размера квадратных матриц в худшем случае

## 4.3 Вывод

В обоих случаях лучшие временные характеристики демонстрирует оптимизированная реализация алгоритма Винограда. Худшие временные характеристики у стандартного алгоритма умножения. Полученные результаты полностью соответствуют трудоемкости описанной формулами 2.1, 2.6, 2.7, 2.12.

# ЗАКЛЮЧЕНИЕ

Целью данной работы являлся анализ алгоритмов умножения матриц: стандартного алгоритма, алгоритма Штрассена и алгоритма Винограда. В ходе лабораторной работы были выполнены следующие задачи:

1) описаны алгоритмы стандартного умножения матриц и алгоритма Винограда;
2) реализованы указанные алгоритмы;
3) проведено тестирование реализаций алгоритмов;
4) проведен анализ временных характеристик реализованных алгоритмов.

По итогам анализа было выявлено, что оптимизированная реализация алгоритма Винограда показывает наилучшие временные характеристики. Худшие временные результаты показывает стандартный алгоритм умножения матриц.

Поставленная цель исследования алгоритмов поиска в массиве была достигнута.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Googletest user's guide. Режим доступа: `https://google.github.io/googletest/`. Дата обращения 8.10.2024.

2. matplotlib. Режим доступа: `https://matplotlib.org/`. Дата обращения 24.10.2024.

3. Демиденко Е. З. *Линейная и нелинейная регресии*. Финансы и статистика, Москва, 1981. 302 с.

4. Strassen V. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

5. D. Coppersmith S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 1(9):251–280, 1990.

6. Мерков А. Б. *Распознавание образов: Построение и обучение вероятностных моделей*. ЛЕНАНД, Москва, 2014. 240 с., ISBN 978-5-9710-0564-3.

7. Роджерс Д. *Алгоритмические основы машинной графики*. Мир, Москва, 1989. 512 с., ISBN 5-03-000476-9.