СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучение алгоритмов умножения матриц на основе стандартного алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда. Также требуется научиться рассчитывать трудоёмкость алгоритмов.

При выполнении лабораторной работы поставлены такие задачи:

1) изучить алгоритмы умножения матриц;

2) оптимизировать алгоритм Винограда;

3) дать теоретическую оценку трудоёмкости стандартного алгоритма, алгоритма Винограда и улучшенного алгоритма Винограда;

4) реализовать три алгоритма;

5) провести их сравнительный анализ.

1 Аналитический раздел

В данном разделе будут рассмотрено умножение матриц, используя стандартный алгоритм и алгоритм Винограда.

Стандартный алгоритм умножения матриц.

Матрицей A размера $m \times n$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. [1] Умножение матриц возможно, только когда число столбцов первой матрицы равно числу строк второй.

Таким образом при умножении матрицы A размерности $m \times n$ на матрицу B размерности $n \times k$ будет матрица C размерностью $m \times k$, где

$$c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} \tag{1.1}$$

Алгоритм Винограда.

При рассмотрении результата умножения матриц видно, что каждый элемент – скалярное произведение векторов, представляющих строки и столбцы матриц. [2]

Рассмотрим 2 вектора $V = (v1, v2, v3, v4)$ и $W = (w1,w2,w3,w4)$. Их скалярное произведение равно:

$$V \bullet W = v1w1 + w2w2 + v3w3 + v4w4 \tag{1.2}$$

Или:

$$V \bullet W = (v1+w2)(v2+w1)+(v3+w4)(v4+w3)-v1v2-v3v4-w1w2-w3w4 \tag{1.3}$$

Можно заметить, что 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. На практике это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Вывод.

Были рассмотрены стандартный алгоритм умножения матриц и алгоритм Винограда, который позволяет предварительно рассчитать некоторые значения, в результате чего уменьшается доля умножений.

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы алгоритмов умножения матриц, введена модель вычисления трудоёмкости и в соответствии с ней подсчитана суммарная трудоёмкость для каждого из 3-х алгоритмов.

Требования к вводу:

— на вход подаются 2 матрицы, и их размерности;

— передаётся результирующая матрица.

Требования к программе:

— корректная обработка умножения матриц размерами 0 на 0;

— корректное умножение двух матриц.

2.1 Схемы алгоритмов

На рисунках 2.1-2.3 представлены схемы алгоритмов умножения матриц.

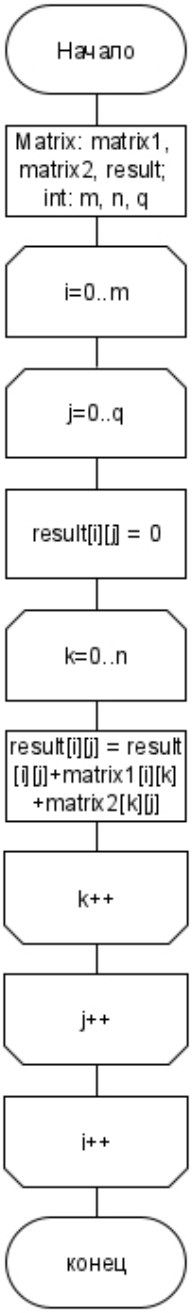

Рисунок 2.1 — Стандартный алгоритм

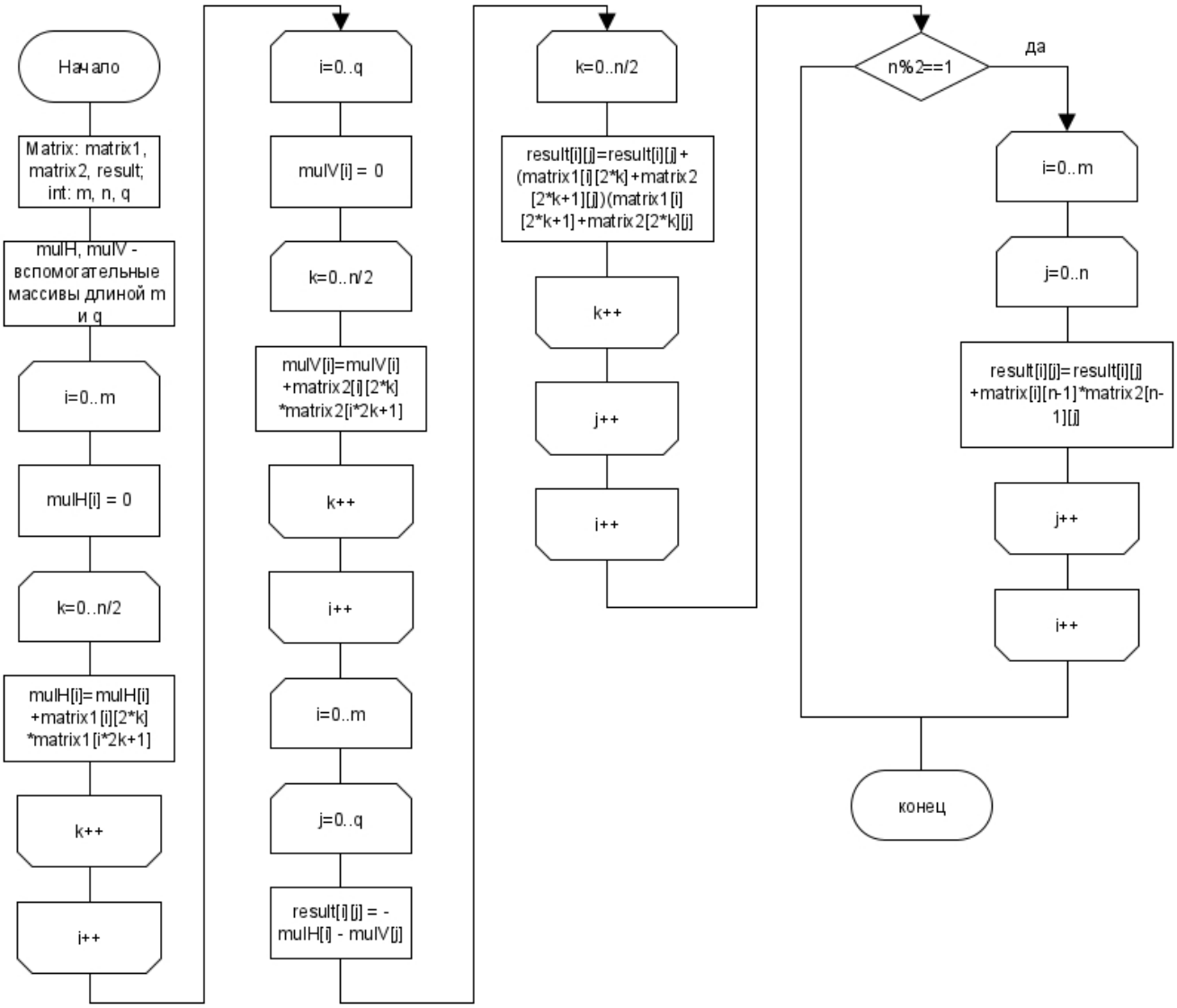

Рисунок 2.2 — Алгоритм Винограда

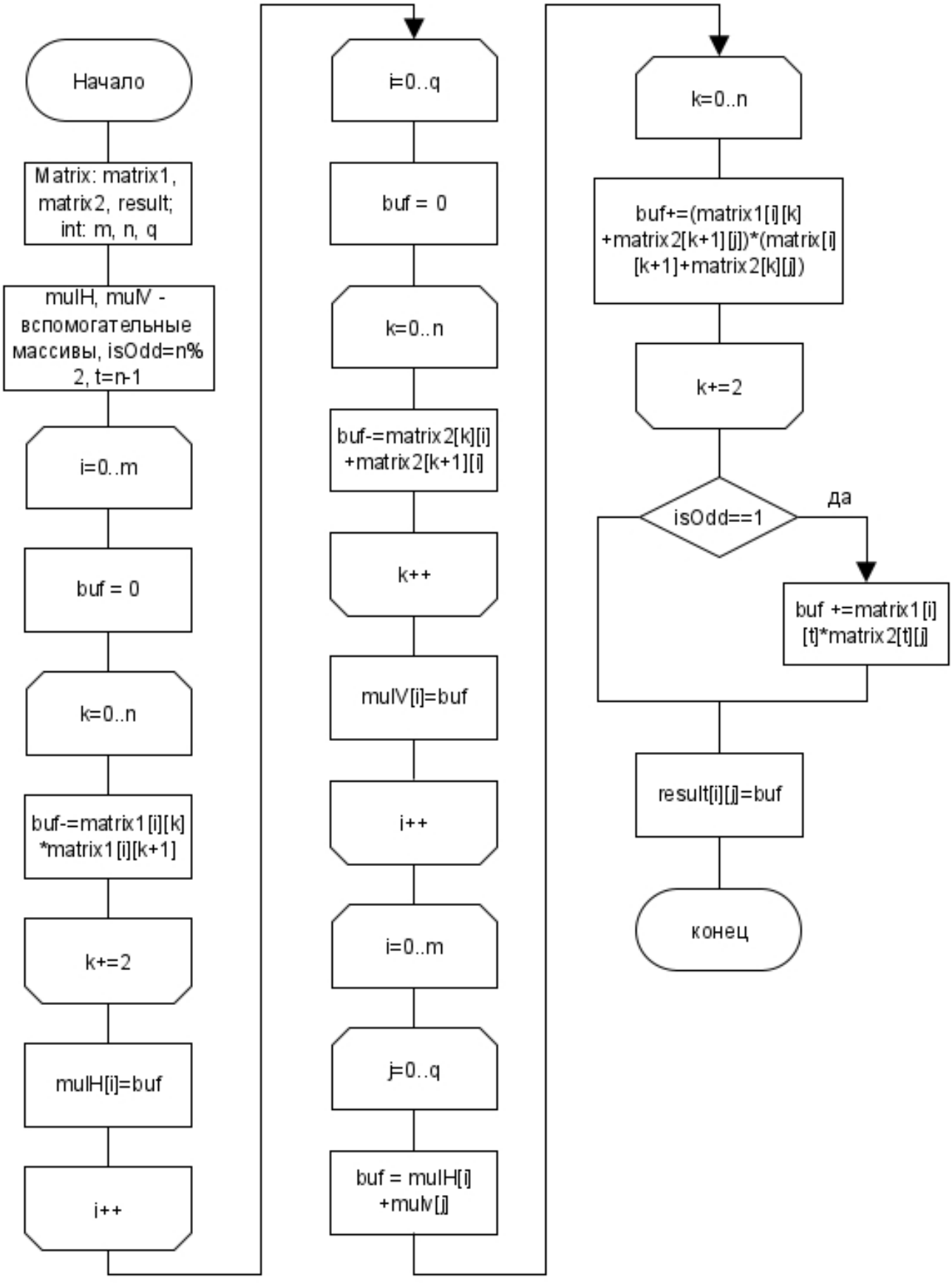

Рисунок 2.3 — Улучшенный алгоритм Винограда

2.2 Трудоёмкость алгоритмов

Введём модель вычислений трудоёмкости:

1) стоимость базовых операций 1: =,+,-,*,==,!=,<,>,<=,>=»%,+=,-=,*=,/=,[]

2) оценка трудоёмкости цикла $f_{for} = f_{init} + f_{comp} + N(f_{body} + f_{inc} + f_{comp})$;

3) оценка трудоёмкости условного оператора, стоимость перехода положим 0, тогда

$$f_{if} = f_{condition} + \begin{cases} min(f_1,f_2) - \text{лучший случай} \\ max(f_1,f_2) - \text{худший случай} \end{cases} \quad (2.1)$$

Классический алгоритм:

$$f_{std} = 2+M*(2+2+Q*(2+3+2+N*(2+8+1+1+1))) = 13MNQ+7MQ+4M+2 \quad (2.2)$$

Алгоритм Винограда:

I Вычисление сумм произведений пар соседних элементов для каждой строки A:

$$f_I = 2 + M*(2+2+3+N/2*(3+1+6+2+3)) = 15/2MN + 7M + 2 \quad (2.3)$$

II Вычисление сумм произведений пар соседних элементов для каждого столбца B аналогично:

$$f_{II} = 15/2QN + 7Q + 2 \quad (2.4)$$

III Заполнение C_{ij}:

$$f_{III} = 2+M(2+2+Q(2+7+3+N/2(3+1+12+5+5))) = 13MNQ+12MQ+4M+2 \quad (2.5)$$

IV Проверка нечётности N:

$$f_{IV} = 2 + \begin{cases} 0 - \text{лучший случай, N-чётное} \\ 13NQ + 4M + 2, - \text{худший случай, N-нечётное} \end{cases} \quad (2.6)$$

Суммарная трудоёмкость:

$$f_{win} = f_I + f_{II} + f_{III} + f_{IV} = 15/2MN + 7M + 2 + 15/2QN + 7Q + 2+$$
$$+ 13MNQ + 12MQ + 4M + 2 + 2 + \begin{cases} 0 \\ 13NQ + 4M + 2 \end{cases} = 13MNQ+$$
$$+ 12MQ + 15/2MN + 15/2QN + 11M + 7Q + 8 + \begin{cases} 0 \\ 13NQ + 4M + 2 \end{cases} \tag{2.7}$$

Улучшенный алгоритм винограда:

I Вычисление сумм произведений пар соседних элементов для каждой строки A:

$$f_I = 2 + M * (2 + 1 + 2 + N/2 * (2 + 4 + 2 + 1) + 2) = 9/2MN + 7M + 2 \tag{2.8}$$

II Вычисление сумм произведений пар соседних элементов для каждого столбца B аналогично:

$$f_{II} = 9/2QN + 7Q + 2 \tag{2.9}$$

III Заполнение C_{ij}:

$$f_{III} = 2 + M * (2 + 2 + Q * (2 + 2 + 1 + 1 + 2 + N/2 * (2 + 8 + 4 + 1 + 1) + 1+$$
$$+ \begin{cases} 0 \\ 4 + 1 + 1 \end{cases} + 2 + 1)) = 8MNQ + MQ * (12 + \begin{cases} 0 \\ 6 \end{cases}) + 4M + 2 \tag{2.10}$$

IV Подсчёт констант: $f_c = 1 + 1 + 1 + 1 = 4$

Суммарная трудоёмкость:

$$f_{winopt} = 9/2MN + 7M + 2 + 9/2QN + 7Q + 2 + 8MNQ +$$
$$+ MQ * (12 + \begin{cases} 0 \\ 6 \end{cases}) + 4M + 2 + 4 =$$
$$= 8MNQ + 9/2MN + 9/2QN + MQ * (12 + \begin{cases} 0 \\ 6 \end{cases}) + 11M + 7Q + 10 \tag{2.11}$$

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель вычисления трудоёмкости, рассчитаны трудоёмкости алгоритмов. Из формул 2.2, 2.7, 2.11 видно, что все три алгоритма зависят от размеров матриц как MNQ, однако, у оптимизированного алгоритма Винограда при этом слагаемом стоит наименьший коэффициент, равный 8, из чего следует, что он должен быть наиболее эффективным.

3 Технологический раздел

В данном разделе будут представлены листинги кода реализованных алгоритмов, указаны использованные оптимизации алгоритма Винограда.

3.1 Средства реализации

В данной работе используется язык программирования C++. Среда разработки Visual Studio Code. Для замера процессорного времени используется функция QueryPerformanceCounter из библиотеки windows.h.

3.2 Листинг кода

В листингах 3.1–3.3 приведены стандартный алгоритм умножения матриц, алгоритм умножения матриц Винограда и оптимизированный алгоритм Винограда.

Листинг 3.1 — Стандартный алгоритм умножения матриц

```
1  void multiplyMatricesStandard(int** matrix1, int m, int n, int** matrix2,
       int q, int** result)
2  {
3      for (int i = 0; i < m; i++)
4      {
5          for (int j = 0; j < q; j++)
6          {
7              result[i][j] = 0;
8              for (int k = 0; k < n; k++)
9              {
10                 result[i][j] = result[i][j] + matrix1[i][k] * matrix2[k][j];
11             }
12         }
13     }
14 }
```

Листинг 3.2 — Алгоритм умножения матриц Винограда

```
1  void multiplyMatricesWinograd(int **matrix1, int m, int n, int **matrix2,
       int q, int **result)
2  {
3      int *mulH = new int[m];
4      for (int i = 0; i < m; i++)
5      {
```

```
6           mulH[i] = 0;
7           for (int k = 0; k < n / 2; k++)
8               mulH[i] = mulH[i] + matrix1[i][2*k] * matrix1[i][2*k + 1];
9       }
10
11      int *mulV = new int[q];
12      for (int i = 0; i < q; i++)
13      {
14          mulV[i] = 0;
15          for (int k = 0; k < n / 2; k++)
16              mulV[i] = mulV[i] + matrix2[2*k][i] * matrix2[2*k+1][i];
17      }
18
19      for (int i = 0; i < m; i++)
20          {
21          for (int j = 0; j < q; j++)
22          {
23              result[i][j] = -mulH[i] - mulV[j];
24              for (int k = 0; k < n/2; k++)
25                  result[i][j] = result[i][j] + (matrix1[i][2 * k] +
                        matrix2[2 * k + 1][j])*(matrix1[i][2 * k + 1] +
                        matrix2[2 * k][j]);
26          }
27      }
28
29      if (n % 2 == 1)
30      {
31          for (int i = 0; i < m; i++)
32              for (int j = 0; j < q; j++)
33                  result[i][j] = result[i][j] + matrix1[i][n - 1] *
                        matrix2[n-1][j];
34      }
35      delete[] mulH;
36      delete[] mulV;
37  }
```

Листинг 3.3 — Улучшенный алгоритм матриц Винограда

```
1  void multiplyMatricesWinogradOptimized(int **matrix1, int m, int n, int
       **matrix2, int q, int **result)
2  {
3      int buf;
4      const int isOdd = n % 2;
5      const int t = n - 1;
6      if (isOdd)
7          n -= 1;
8
```

```
 9
10      int *mulH = new int[m];
11      for (int i = 0; i < m; i++)
12      {
13          buf = 0;
14          for (int k = 0; k < n; k += 2)
15              buf -= matrix1[i][k] * matrix1[i][k + 1];
16          mulH[i] = buf;
17      }
18
19      int *mulV = new int[q];
20      for (int i = 0; i < q; i++)
21      {
22          buf = 0;
23          for (int k = 0; k < n; k += 2)
24              buf -= matrix2[k][i] * matrix2[k+1][i];
25          mulV[i] = buf;
26      }
27
28      for (int i = 0; i < m; i++)
29      {
30          for (int j = 0; j < q; j++)
31          {
32              buf = mulH[i] + mulV[j];
33              for (int k = 0; k < n; k += 2)
34                  buf += (matrix1[i][k] + matrix2[k + 1][j]) * (matrix1[i][k
                        + 1] + matrix2[k][j]);
35              if (isOdd==1)
36                  buf += matrix1[i][t] * matrix2[t][j];
37              result[i][j] = buf;
38          }
39      }
40      delete[] mulH;
41      delete[] mulV;
42  }
```

3.2.1 Оптимизация алгоритма Винограда

1. Заранее вычисляются такие константы как isOdd=n%2 и t=n-1.

2. Накопление результата в буфер, а после помещение буфера в ячейку.

3. Где это возможно используются -= и +=.

4. Вместо n/2 цикл идёт до n с шагом 2, что представлено в листинге 3.4.

Листинг 3.4 — Оптимизации алгоритма Винограда №2–4

```
1       for (int i = 0; i < m; i++)
2       {
3           buf = 0;
4           for (int k = 0; k < n; k += 2)
5               buf -= matrix1[i][k] * matrix1[i][k + 1];
6           mulH[i] = buf;
7       }
```

5. Объединены части 3 и 4 алгоритма Винограда, как это отображено в листинге 3.5

Листинг 3.5 — Оптимизация алгоритма Винограда №5

```
1       for (int i = 0; i < m; i++)
2       {
3           for (int j = 0; j < q; j++)
4           {
5               buf = mulH[i] + mulV[j];
6               for (int k = 0; k < n; k += 2)
7                   buf += (matrix1[i][k] + matrix2[k + 1][j]) * (matrix1[i][k
                        + 1] + matrix2[k][j]);
8               if (isOdd==1)
9                   buf += matrix1[i][t] * matrix2[t][j];
10              result[i][j] = buf;
11          }
12      }
```

3.3 Проведение тестирования

Изначально было проведено тестирование стандартного алгоритма умножения матриц на квадратных матрицах размерами $0 \times 0, 1 \times 1, 3 \times 3, 6 \times 6$ и прямоугольных $1 \times 2 \bullet 2 \times 1, 2 \times 4 \bullet 4 \times 2$ и $3 \times 5 \bullet 5 \times 5$, все тесты были пройдены успешно.

После чего была проведена серия тестов на матрицах $A = M \times Q$ и $B = Q \times N$, где M, N, Q принимают значения от 1 до 10, независимо друг от друга, матрицы заполняются случайными значениями. Сначала выполнялось умножение стандартным алгоритмом, результат которого принимался за контрольное значение, после чего результаты алгоритма Винограда и оптимизированного алгоритма Винограда сравнивались с контрольным значением и

при совпадении результат считался корректным. Все тесты были пройдены успешно.

Вывод.

В данном разделе были рассмотрены листинги кода реализованных алгоритмов, а также их тестирование.

4 Исследовательский раздел

В данном разделе будет измерено время работы алгоритмов и сделаны выводы на основе полученных данных.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Для каждого из алгоритмов был проведён замер времени на матрицах размером от 100 до 1000 с шагом 100. Для каждого размера было проведено 10 повторений.

На рисунке 4.1 представлено сравнение времени для алгоритмов при чётных размерах матриц.

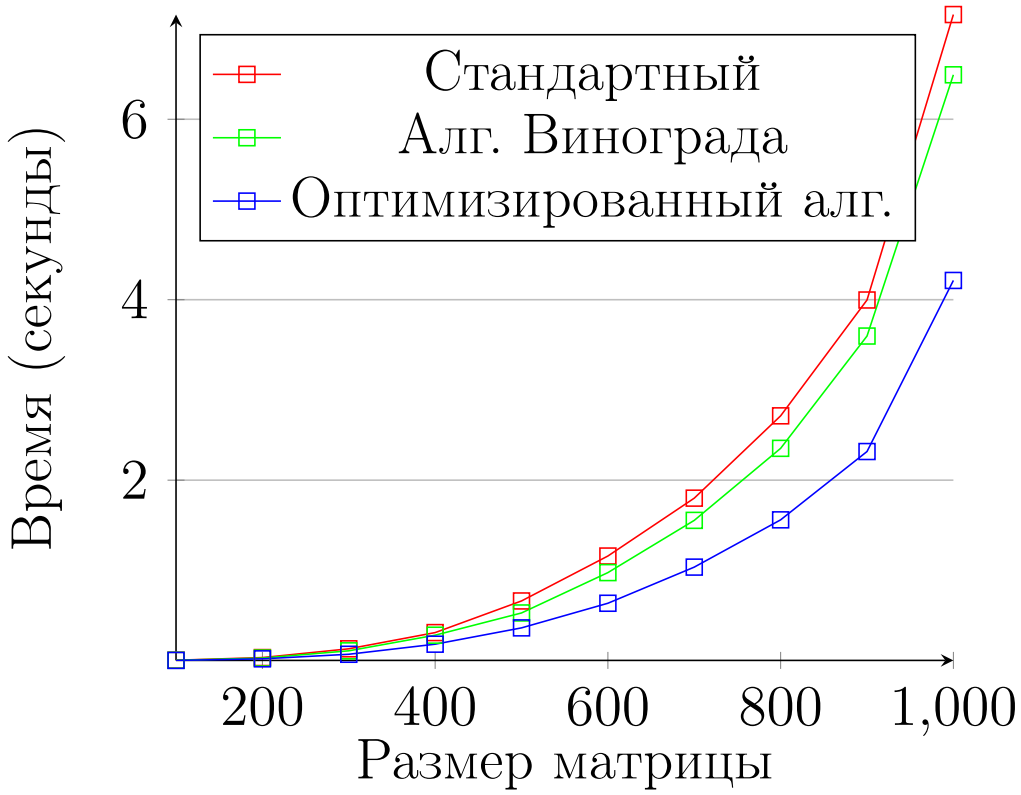

Рисунок 4.1 — Сравнение времени работы алгоритмов на матрицах чётного размера

На рисунке 4.2 представлено сравнение времени работы для алгоритмов при нечётных размерах матриц.

Из рисунков 4.2 и 4.1 видно, что для всех трёх алгоритмов асимптотика роста общая, однако как для чётных, так и для нечётных матриц оптимизированный алгоритм Винограда является самым быстрым, следующий по скорости алгоритм Винограда и самый медленный – стандартный алгоритм умножения матриц.

Таким образом, данные, полученные в результате проведённых экспериментов подтверждают корректность рассчитанных ранее трудоёмкостей алгоритмов.

Вывод.

В итоге, можно сказать о том, что оптимизированный алгоритм Винограда является самым эффективным из рассмотренных.

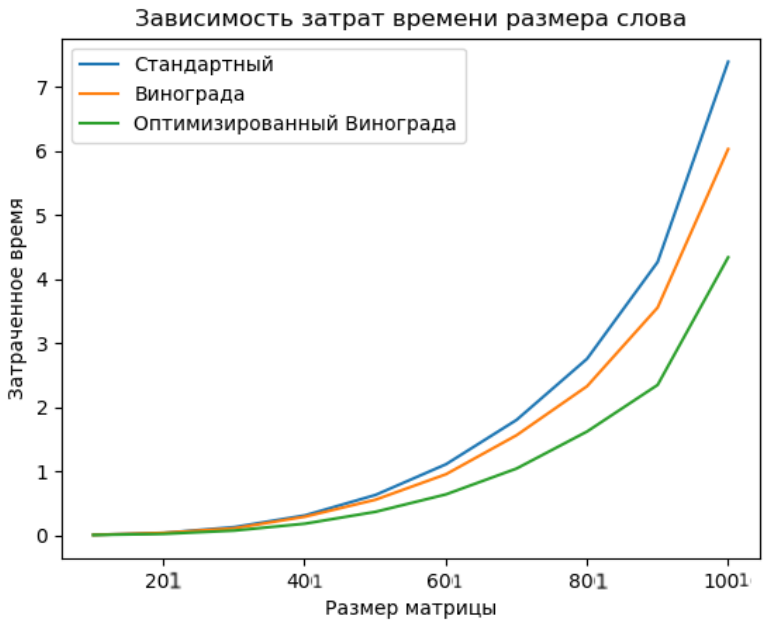

Рисунок 4.2 — Сравнение времени работы алгоритмов на матрицах нечётного размера

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы цель была достигнута – были изучены алгоритмы умножения матриц. Все поставленные задачи были выполнены: оптимизирован алгоритм Винограда, дана теоретическая оценка трудоёмкости стандартного алгоритма, алгоритма Винограда и улучшенного алгоритма Винограда, все они были реализованы и проведён их сравнительный анализ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. И.В. Белоусов. Матрицы и определители: учебное пособие по линейной алгебре. — 2006. — 101 с.

2. Умножение матриц[Электронный ресурс]. — 2005. — Режим доступа: `http://algolib.narod.ru/Math/Matrix.html` (дата обращения: 29.09.2020).