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»

Отчет по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Умножение матриц

Студент Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) ____________

Преподаватель Строганов Д.В.

Москва — 2023 г.

Оглавление

Введение

Алгоритм Копперсмита — Винограда является алгоритмом умножения матриц и был разработан с целью снижения сложности этой операции. Он основан на использовании техники предварительных вычислений, которая позволяет уменьшить количество операций умножения.

Алгоритм Копперсмита — Винограда, впервые предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом, имеет асимптотическую сложность $O(n^{2,3755})$, где n - размер стороны матрицы.

Основная идея алгоритма заключается в том, чтобы предварительно вычислить некоторые значения, которые будут использоваться при последующем умножении матриц. Алгоритм состоит из следующих шагов:

1. Инициализация: создание матриц для хранения предварительных вычислений и определение размеров матриц, которые будут умножаться.

2. Предварительные вычисления: для каждой строки первой матрицы и каждого столбца второй матрицы вычисляются некоторые промежуточные значения, которые будут использоваться при финальном умножении. Эти значения сохраняются в отдельных матрицах.

3. Умножение: используя предварительно вычисленные значения, производится финальное умножение матриц.

4. Возврат результата: полученная после умножения матрица возвращается как результат операции.

Алгоритм Копперсмита — Винограда имеет лучшую асимптотическую сложность по сравнению с другими известными алгоритмами умножения матриц, однако он не используется в практике из-за высокой константы пропорциональности. Он становится эффективным только для матриц большого размера, которые превышают память современных компьютеров.

1 Аналитическая часть

1.1 Цель и задачи

Целью данной лабораторной работы является: Изучение алгоритмов умножения матриц Копперсмита-Винограда, оптимизированного Копперсмита-Винограда. Для достижения цели следует поставить следующие задачи:

1. Реализация трёх алгоритмов умножения матриц: обычный, Копперсмита-Винограда, оптимизированный Копперсмита-Винограда.
2. Выполнить замеры процессорного времени работы реализаций алгоритмов;
3. Сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений.
4. Сравнительный анализ алгоритмов на основе экспериментальных данных.

1.2 Стандартный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.3}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.3 Алгоритм 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 равно: $V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления бо́льшего количества операций, чем стандартный алгоритм умножения матриц, оно предлагает возможность предварительной обработки. Это означает, что некоторые значения могут быть вычислены заранее и сохранены для последующего использования. В результате каждый элемент матрицы будет получен с помощью только двух умножений и пяти сложений, а также сложения с предварительно посчитанными суммами соседних элементов текущих строк и столбцов.

Таким образом, алгоритм Копперсмита — Винограда может быть более эффективным на практике, поскольку операция сложения выполняется быстрее операции умножения в современных компьютерах. Это позволяет сократить общее количество операций и уменьшить время выполнения алгоритма.

1.4 Оптимизированный алгоритм Копперсмита – Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- Используется битовый сдвиг вместо умножения на 2
- Заменена операция x = x + k на x += k
- Предвычисляются слагаемые для алгоритма.

Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда: обычный и оптимизированный.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов умножения матриц: обычный, алгоритм Копперсмита – Винограда, оптимизированный алгоритм Копперсмита – Винограда. Проведен анализ трудоемкости данных алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема алгоритма обычного нахождения матриц. На рисунке 2.2 приведена схема алгоритма Копперсмита – Винограда нахождения матриц. На рисунке 2.3 приведена схема алгоритма оптимизированного Копперсмита – Винограда нахождения матриц.

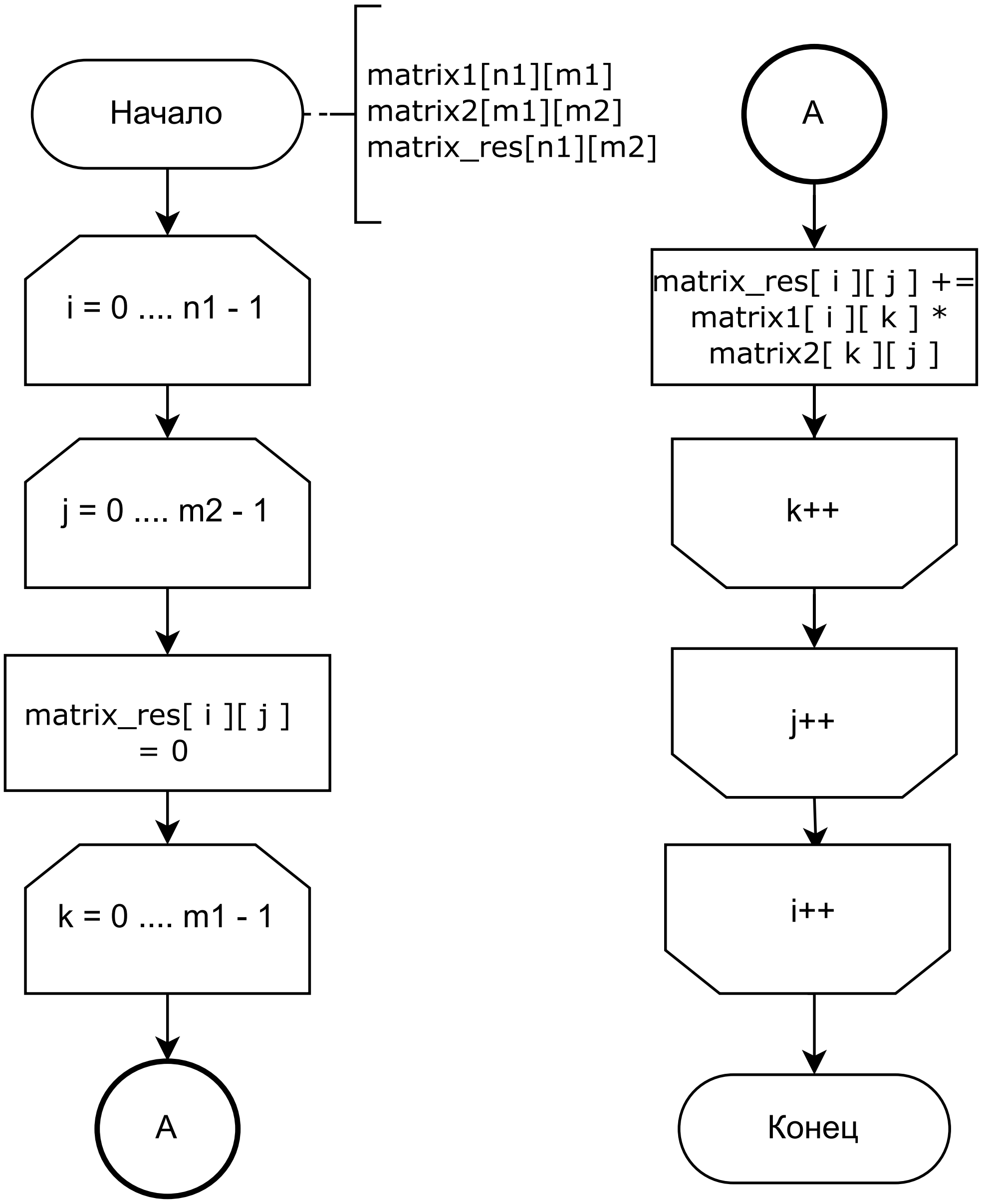

Рисунок 2.1 – Схема матричного алгоритма нахождения расстояния Левенштейна

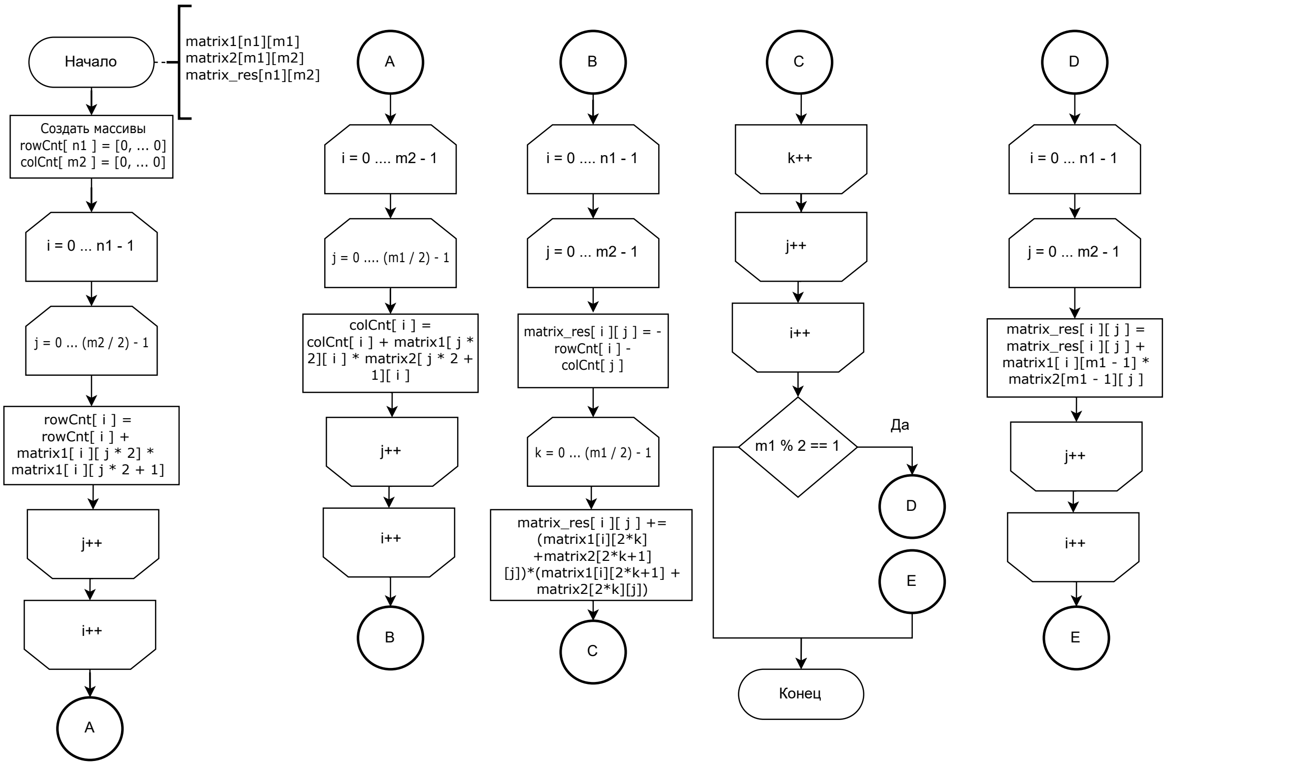

Рисунок 2.2 – схема алгоритма Копперсмита – Винограда нахождения матриц

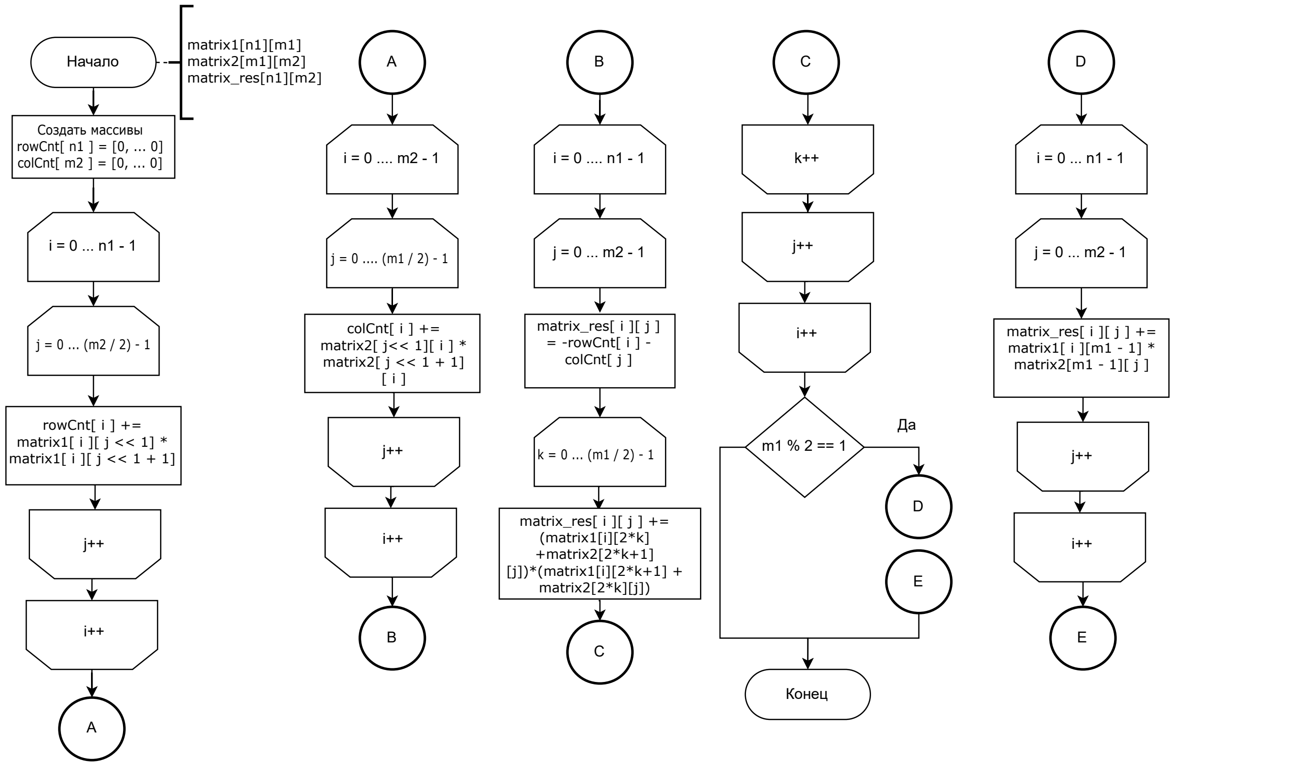

Рисунок 2.3 – схема алгоритма оптимизированного Копперсмита – Винограда нахождения матриц

2.2 Трудоемкость алгоритмов

Трудоемкость рассчитывается следующим образом:

1. Операции из следующего списка имеют трудоемкость 1.

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

   Операции из следующего списка имеют трудоемкость 2.

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

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

3. Трудоемкость цикла рассчитывается, как:

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

   где N - количество итераций цикла.

Представим в вычислениях:

Размер первой матрицы – (n_1, m_1), размер второй матрицы – (m_1, m_2).

2.2.1 Обычный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма рассчитывается как:

$$\begin{aligned} f_{basic\_alg} = 2 + n_1(2 + 2 + m_2(2 + 2 + m_1(2 + 10))) = \\ 12m_1m_2n_1 + 4m_2n_1 + 4n_1 + 2. \end{aligned} \tag{2.5}$$

2.2.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда является суммой трудоёмкостей следующих действий:

1. Заполнения массива rowCnt:

$$f_{rowCnt} = 2 + n_1(2 + 2 + \frac{m_1}{2}(2 + 12)) = 7m_1n_1 + 4n_1 + 2. \tag{2.6}$$

2. Заполнения массива colCnt:

$$f_{colCnt} = 2 + m_2(2 + 2 + \frac{m_1}{2}(2 + 12)) = 7m_1m_2 + 4n_1 + 2. \tag{2.7}$$

3. Основного цикла заполнения матрицы:

$$\begin{gathered} f_{cycle} = 2 + n_1(2 + 2 + m_2(2 + 2 + 7 + \frac{m_1}{2}(2 + 23))) = \\ 12.5m_1cn_1 + 11m_2n_1 + 4n_1 + 2. \end{gathered} \tag{2.8}$$

4. Цикла для дополнения умножения, если m_1 нечётный:

$$f_{last} = \begin{cases} 2, & m_1 \text{ чётный}, \\ 2 + 2 + n_1(2 + 2 + c(2 + 13)) = 15m_2n_1 + 4n_1 + 4, & \text{иначе}. \end{cases} \tag{2.9}$$

Итак, для лучшего случая (m_1 чётный):

$$\begin{aligned} f_{vin\_b} = 7m_1n_1 + 4n_1 + 2 + 7m_1m_2 + 4n_1 \\ +2 + 12.5m_1m_2n_1 + 11m_2n_1 + 4n_1 + 2 + 2 = \\ 12.5m_1m_2n_1 + 7m_1n_1 + 7m_1m_2 + 11m_2n_1 + 12n_1 + 8. \end{aligned} \tag{2.10}$$

Для худшего случая (m_1 нечётный):

$$\begin{aligned} f_{vin\_w} = 7m_1n_1 + 4n_1 + 2 + 7m_1m_2 + 4n_1 \\ +2 + 12.5m_1m_2n_1 + 11m_2n_1 + 4n_1 + 2 + 15m_2n_1 + 4n_1 + 4 = \\ = 12.5m_1m_2n_1 + 7m_1n_1 + 7m_1c + 26m_2n_1 + 16n_1 + 10. \end{aligned} \tag{2.11}$$

2.2.3 Оптимизированный алгоритм Винограда

Трудоёмкость оптимизированного алгоритма Винограда является суммой трудоёмкостей следующих последовательно выполненных действий.

1. Заполнения массива rowCnt:

$$f_{rowCnt} = 2 + n_1(2 + 2 + \frac{m_1}{2}(3 + 12)) = 7.5m_1n_1 + 4n_1 + 2. \qquad (2.12)$$

2. Заполнения массива colCnt:

$$f_{colCnt} = 2 + m_2(2 + 2 + \frac{m_1}{2}(3 + 12)) = 7.5m_1m_2 + 4n_1 + 2. \qquad (2.13)$$

3. Основного цикла заполнения матрицы:

$$\begin{aligned} f_{cycle} = 2 + n_1(2 + 2 + m_2(2 + 2 + 7 + \frac{m_1}{2}(2 + 23))) = \\ 12.5m_1m_2n_1 + 11m_2n_1 + 4n_1 + 2. \end{aligned} \qquad (2.14)$$

4. Цикла для дополнения умножения, если m_1 нечётный:

$$f_{last} = \begin{cases} 2, & m_1 \text{ чётный}, \\ 2 + 2 + n_1(2 + 2 + m_2(2 + 13)) = \\ \qquad 15m_2n_1 + 4n_1 + 4, \text{иначе.} \end{cases} \qquad (2.15)$$

Итак, для лучшего случая (m_1 чётный):

$$\begin{aligned} f_{vinOpt\_b} = 7.5m_1n_1 + 5n_1 + 5 + 7.5m_1m_2 + 5n_1 \\ +2 + 9m_1m_2n_1 + 12m_2n_1 + 4n_1 + 5 = \\ 9m_1m_2n_1 + 7.5m_1n_1 + 7.5m_1m_2 + 12m_2n_1 + 14n_1 + 12. \end{aligned} \qquad (2.16)$$

Для худшего случая (m_1 нечётный):

$$\begin{aligned} f_{vinOpt\_w} = 7.5m_1n_1 + 5n_1 + 5 + 7.5m_1m_2 + 5n_1 \\ +2 + 9m_1m_2n_1 + 12m_2n_1 + 9m_2n_1 + 4n_1 + 5 = \\ 9m_1m_2n_1 + 7.5m_1n_1 + 7.5m_1m_2 + 21m_2n_1 + 14n_1 + 12. \end{aligned} \qquad (2.17)$$

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов, были вычислены трудоемкость стандартного алгоритма умножения матриц 2.5, алгоритмов Винограда стандартного и оптимизированного соответственно в худших 2.11, 2.17 и лучших 2.10, 2.16 случаях. Исходя из результатов анализа трудоемкости оптимизированный алгоритм Винограда должен работать быстрее стандартного, алгоритм стандартного умножения матриц будет работать медленней всего.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к вводу

К вводу программы прилагаются данные требования:

1. Перед вводом матрицы запрашиваются ее размерности.
2. На вход подаются две матрицы.
3. Ввод матрицы - числа типа int.

3.2 Требования к программе

К программе прилагаются данные требования:

В вводе размерностей (n1, n2, m1, m2) n2 обязана равняться m1. На выход программа должна вывести три итоговые матрицы, рассчитанные разными методами.

3.3 Требования к программному обеспечению

К программе предъявляется ряд требований:

- у пользователя есть выбор алгоритма, или какой-то один, или все сразу;
- на вход подаются две матрицы с содержанием – числа типа int;
- на выходе — матрица с результатом умножения исходных матриц.

3.4 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C. Выбор обусловлен наличием библиотек для измерения времени, наличием инструментов для работы с матрицами.

3.5 Сведения о модулях программы

Программа состоит из следующих модулей: main.c - главный файл программы, в котором располагается вся программа, time.c - файл программы с замерами времени.

3.6 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3, приведены реализации алгоритмов умножения матриц.

Листинг 3.1 – Стандартный алгоритм умножения матриц.

```
int multMatrix(int **matrix1, int **matrix2, int
    **matrix_res, size_t n1, size_t m1, size_t m2){
    for (size_t i = 0; i < n1; i++) {
        for (size_t j = 0; j < m2; j++) {
            matrix_res[i][j] = 0;
            for (size_t k = 0; k < m1; k++) {
                matrix_res[i][j] += matrix1[i][k] *
                    matrix2[k][j];
            }
        }
    }
    return 0;
}
```

Листинг 3.2 – Алгоритм Винограда

```
int Vinograd(int **matrix1, int **matrix2, int **matrix_res,
    size_t n1, size_t m1, size_t m2)
{
    int *rowCnt = (int*)malloc(n1 * sizeof(int));
    int *colCnt = (int*)malloc(m2 * sizeof(int));;
    for (size_t i = 0; i < n1; i++)
    {
        rowCnt[i] = 0;
        for (size_t j = 0; j < m1 / 2; j++)
        rowCnt[i] = rowCnt[i] + matrix1[i][j * 2] *
            matrix1[i][(j*2) + 1];
    }
    for (size_t i = 0; i < m2; i++) {
        colCnt[i] = 0;
        for (size_t j = 0; j < m1 / 2; j++)
        colCnt[i] = colCnt[i] + matrix2[j * 2][i] * matrix2[j*2 +
            1][i];
    }

    for (size_t i = 0; i < n1; i++) {
        for (size_t j = 0; j < m2; j++) {
            matrix_res[i][j] = -rowCnt[i] - colCnt[j];
            for (size_t k = 0; k < m1 / 2; k++) {
                matrix_res[i][j] += (matrix1[i][2*k] +
                    matrix2[2*k+1][j]) * \
                (matrix1[i][2*k+1] + matrix2[2*k][j]);
            }
        }
    }
    if (m1 % 2 == 1){
        for (size_t i = 0; i < n1; i++)
            for (size_t j = 0; j < m2; j++)
                matrix_res[i][j] = matrix_res[i][j] +
                    matrix1[i][m1 - 1] * matrix2[m1 - 1][j];
    }
    free(rowCnt);
    free(colCnt);
    return 0;
}
```

Листинг 3.3 – Оптимизированный Алгоритм Винограда

```
int Vinograd_Opt(int **matrix1, int **matrix2, int
    **matrix_res, size_t n1, size_t m1, size_t m2)
{
    int *rowCnt = (int*)malloc(n1 * sizeof(int));
    int *colCnt = (int*)malloc(m2 * sizeof(int));;
    for (size_t i = 0; i < n1; i++) {
        rowCnt[i] = 0;
        for (int j = 0; j < m1 / 2; j++)
            rowCnt[i] += matrix1[i][j << 1] *
                matrix1[i][(j<<1) + 1];
    }
    for (size_t i = 0; i < m2; i++) {
        colCnt[i] = 0;
        for (int j = 0; j < m1 / 2; j++)
            colCnt[i] += matrix2[j * 2][i] * matrix2[(j<<1) +
                1][i];
    }
    for (size_t i = 0; i < n1; i++) {
        for (size_t j = 0; j < m2; j++) {
            matrix_res[i][j] = -rowCnt[i] - colCnt[j];
            for (int k = 0; k < m1 / 2; k++) {
                matrix_res[i][j] += (matrix1[i][k<<1] +
                    matrix2[(k<<1)+1][j]) * \
                (matrix1[i][2*k+1] + matrix2[2*k][j]);
            }
        }
    }
    if (m1 % 2 == 1) {
        for (size_t i = 0; i < n1; i++)
        for (size_t j = 0; j < m2; j++)
        matrix_res[i][j] += matrix1[i][m1 - 1] * matrix2[m1 -
            1][j];
    }
    free(rowCnt);
    free(colCnt);
    return 0;
}
```

3.7 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц.

Таблица 3.1 – Тестирование функций

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}30 & 36 & 42\\ 66 & 81 & 96\\ 102 & 126 & 150\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$	$\begin{pmatrix}22 & 28\\ 49 & 64\end{pmatrix}$
$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$
$\begin{pmatrix}1 & -2 & 3\\ 4 & 5 & -6\\ -7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & -2 & 3\\ 4 & 5 & -6\\ -7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}-28 & 12 & 42\\ 66 & -31 & -72\\ -38 & 126 & 12\end{pmatrix}$
$\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}22 & 28\\ 49 & 64\end{pmatrix}$
$\begin{pmatrix}0 & 1\\ 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 1\\ 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 1\\ 0 & 1\end{pmatrix}$
$\begin{pmatrix}4\\ 4\\ 4\end{pmatrix}$	$\begin{pmatrix}-4 & -4 & -4\end{pmatrix}$	$\begin{pmatrix}-48\end{pmatrix}$
$\begin{pmatrix}20 & 27\\ 19 & 16\end{pmatrix}$	$\begin{pmatrix}-2 & -3\\ -5 & -7\end{pmatrix}$	$\begin{pmatrix}-175 & -249\\ -118 & -169\end{pmatrix}$
$\begin{pmatrix}0\end{pmatrix}$	$\begin{pmatrix}0\end{pmatrix}$	$\begin{pmatrix}0\end{pmatrix}$

Все алгоритмы прошли проверку.

Вывод

Были разработаны и протестированы алгоритмы: стандартный алгоритм умножения матриц, Алгоритм Винограда, оптимизированный Алгоритм Винограда.

4 Исследовательская часть

В данном разделе будут приведены постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: 64-разрядная операционная система, процессор x64.
- Память: 16 Гб.
- Процессор: Intel(R) Core(TM) i7–4700HQ CPU @ 2.40 ГГц.

Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Расчет затрат памяти

Пусть (n1, m1) – размеры первой матрицы, (m1, m2) – размеры второй матрицы.

Стандартный алгоритм перемножения матриц:

– Матрица 1 – n1 * m1 * sizeof(int);

– Матрица 2 – m1 * m2 * sizeof(int);

– Переменные с размером матриц – 3 * sizeof(int).

Алгоритм Винограда, оптимизированный алгоритм Винограда перемножения матриц:

– Матрица 1 – n1 * m1 * sizeof(int);

– Матрица 2 – m1 * m2 * sizeof(int);

– Переменные с размером матриц – 3 * sizeof(int);

– Вектор RowCnt – n1 * sizeof(int);

– Вектор ColCnt – m2 * sizeof(int);

В итоге, алгоритм Винограда занимает больше памяти из-за использования дополнительных векторов при вычислении.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции clock() из библиотеки <time.h> языка С. Данная функция возвращает реальное время с момента инициализации текущего процессора, типа long в секундах.

Замеры времени для каждого перемножения матриц проводились 100 раз. В качестве результата взято среднее время работы алгоритма.

Результаты замеров при четных размерах матриц приведены в таблице 4.1 (время в мс.).

Результаты замеров при нечетных размерах матриц приведены в таблице 4.2 (время в мс.).

Таблица 4.1 – Результаты замеров времени при четных размерах(мс.)

Размерность матриц	Стандартный алгоритм	Алгоритм Винограда	Оптимизированный Алгоритм Винограда
50x50	0.000630	0.000460	0.000370
100x100	0.004320	0.003520	0.003130
150x150	0.013970	0.011710	0.010700
200x200	0.033100	0.027740	0.025190
250x250	0.065670	0.056320	0.050710
300x300	0.123230	0.104490	0.093440

Таблица 4.2 – Результаты замеров времени при нечетных размерах(мс.)

Размерность матриц	Стандартный алгоритм	Алгоритм Винограда	Оптимизированный Алгоритм Винограда
51x51	0.000550	0.000510	0.000450
101x101	0.004420	0.003780	0.003360
151x151	0.014270	0.012070	0.010940
201x201	0.033960	0.028630	0.026120
251x251	0.066540	0.057090	0.051110
301x301	0.125730	0.109090	0.097860

Вывод

Алгоритм Копперсмита - Винограда имеет меньшую трудоемкость и выполняется в среднем в 1.3 раза быстрее, чем обычный алгоритм умножения матриц. Учлучшенный алгоритм Копперсмита - Винограда работает в 1.5 раз быстрее стандартного умножения матриц и в 1.2 быстрее оригинального алгоритма Винограда.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

1. Были изучены и реализованы 3 алгоритма перемножения матриц: обычный, Копперсмита – Винограда, оптимизированный Копперсмита – Винограда.

2. Был произведён анализ трудоёмкости алгоритмов;

3. Выполнены замеры процессорного времени работы реализаций алгоритмов;

4. Был сделан сравнительный анализ алгоритмов на основе экспериментальных данных.

Можно сделать вывод, что выбор алгоритма умножения матриц зависит от размера матриц и требуемой временной эффективности. Алгоритм Винограда занимает больше памяти, чем стандартный алгоритм перемножения из-за использования дополнительных векторов при вычислении. Алгоритм Копперсмита - Винограда имеет меньшую трудоемкость и выполняется в среднем в 1.3 раза быстрее, чем обычный алгоритм умножения матриц. Улучшенный алгоритм Копперсмита - Винограда работает в 1.5 раз быстрее стандартного умножения матриц и в 1.2 быстрее оригинального алгоритма Винограда.

Список использованных источников

[1] Беллман Р. Введение в теорию матриц. — М.: Мир, 1969. С. 38 — 42.

[2] Coppersmith D. and Winograd S. Matrix multiplication via arithmetic progressions. — Ж.: Journal of Symbolic Computation 9 1990. С. 251—280

[3] Кормен Т. Алгоритмы: построение и анализ [Текст] / Кормен Т. - Вильямс, 2014. - 198 с. - 219 с.