Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 2

ТЕМА Исследование работы алгоритма Винограда для умножения матриц

Студент группы ИУ7-54,
Лозовский Алексей

2019 г.

Содержание

Введение

На сегодняшний день матрица, как математический объект, занимает все более значимую позицию в различных сферах жизни человека, начиная от машинной графики и заканчивая решением задач в области физики, математики, экономики и т.д. Посчитать матрицу размерами 3x3 или 5x5 не должно составить труда, но что если матрица состоит из большего числа значений? Для таких случаев был разработан алгоритм, поволяющий обрабатывать матрицы любой размерности. В зависимости от количества поданных данных, меняется и время работы алгоритма, поэтому со временем были разработаны алгоритмы, способные более эффективным путем выполнять поставленную человеком задачу.

Для чего же простому человеку может понадобиться вычисление матриц? Прежде всего в личных целях, проверить себя, при решении задачи в школе или университете. Кроме того, матрицы широко применяются

- В математике для компактной записи систем линейных алгебраических или дифференциальных уравнений, как это подробно описано в [1];
- при построении экономических моделей, подробнее см. в [2]

1 Аналитическая часть

1.1 Описание алгоритмов

Согласно [3] и [4] **умножение матриц** — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Пусть даны две прямоугольные матрицы A, B размерности $l \times m$ и $m \times n$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

Тогда матрица C размерностью $l \times n$:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix}$$

вычисляется по формуле (1):

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = 1, 2, \ldots, k; j = 1, 2, \ldots, n) \tag{1}$$

Для увеличения скорости выполнения умножения матриц был разработан алгоритм Винограда, согласно [5] идея алгоритма заключается в том, чтобы заранее вычислять некоторые значения для дальнейшего использования впоследствии.

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

$$V * W = v1w1 + v2w2 + v3w3 + v4w4 \tag{2}$$

Выражение (2) можно переписать, как (3):

$$V * W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \tag{3}$$

Выражение (3) допускает предварительную обработку $v1v2, v3v4, w1w2, w3w4$. Их можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй.

Тогда $_{ij}$ вычисляется как (4):

$$\begin{aligned} c_{ij} = (a_{i1} + b_{2j}) * (a_{i2} + b_{1j}) + \cdots + (a_{i(m-1)} + b_{mj}) * (a_{im} b_{(m-1)j}) - \\ -(a_{i1} * a_{i2}) - \cdots - (a_{i(m-1)} * a_{im}) - (b_{1j} * b_{2j}) - \cdots - (b_{(m-1)j} b_{mj}) \\ (i = 1, 2, \ldots, l; j = 1, 2, \ldots, n) \end{aligned} \tag{1}$$

1.2 Задание на выполнение лабораторной работы

- Реализовать стандартный алгоритм умножения матриц;
- Реализовать и опитимизировать алгоритм Винограда;
- Проанализировать работу каждого из алгоритмов;

2 Конструкторская часть

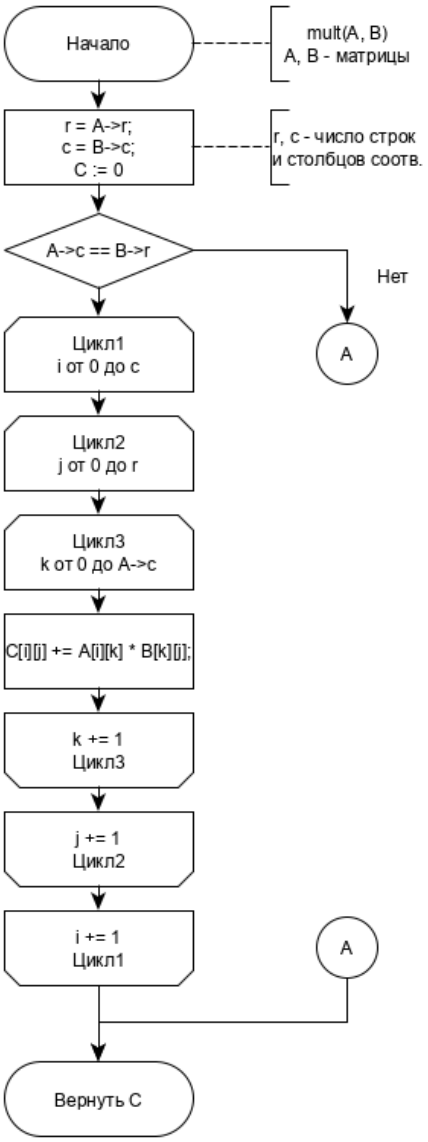
2.1 Структура кода и представление данных

Для написания кода был выбран структурный стиль программирования. Матрица представлена структурой, хранящей ее описание(количество строк и столбцов). Работа будет осуществляться с целочисленными значениями типа int.

2.2 Разработка алгоритмов

Тип оператора	Стоимость
Арифметические операторы	1
Операторы сравнения	1
Логические операторы	1
Побитовые операторы	1
Составное присваивание	1
Операторы работы с указателями	1
Другие операторы	1
функция swap(x,y)	2

2.2.1 Стандартный алгоритм



$$F_1 = 3$$

$$F_{cond} + 1 + \begin{cases} F_{cbody}, & c == r \\ 0, & \text{иначе} \end{cases}$$

$$F_{outer} = 2 + BC * (1 + F_{middle}$$

$$F_{middle} = 2 + R * (1 + F_{inner}$$

$$F_{inner} = 2 + AC * (2 + F_{body}$$

$$F_{body} = 6 + 1 + 1+$$

$$+1)+$$

$$+1)+$$

$$+1)$$

Рис. 1: Стандартный алгоритм

Оператор	Количество
F_1	
=	3
F_{cond}	
<	1
F_{body}	
[]	BC*R*AC*6
*	BC*R*AC
+=	BC*R*AC
<	BC*R*AC
F_{inner}	
<	R*BC
+=	R*BC
F_{outer}	
<	BC
+=	BC

2.2.2 Алгоритм Винограда

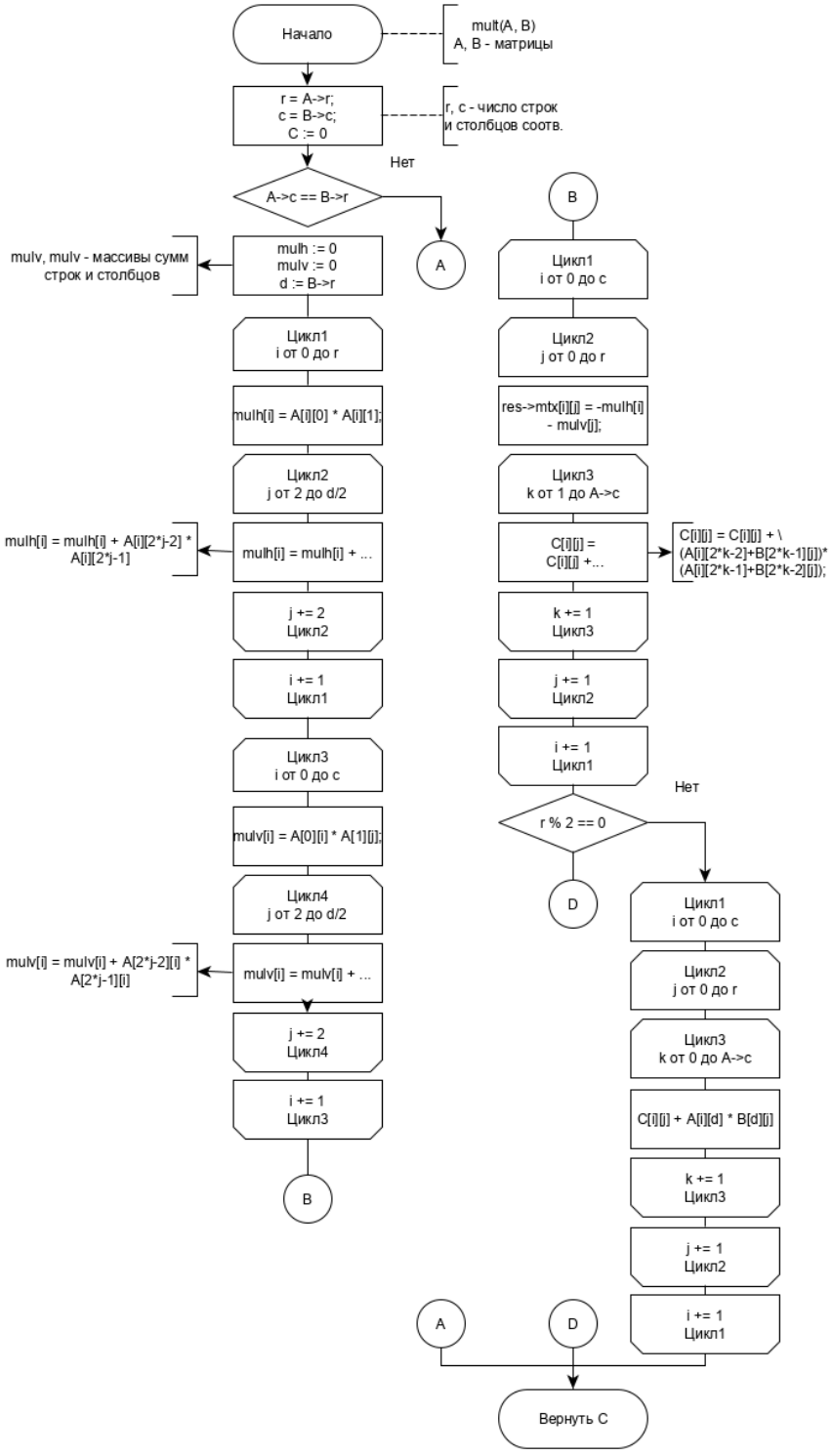


Рис. 2: Алгоритм Винограда

2.2.3 Оптимизация Алгоритма Винограда

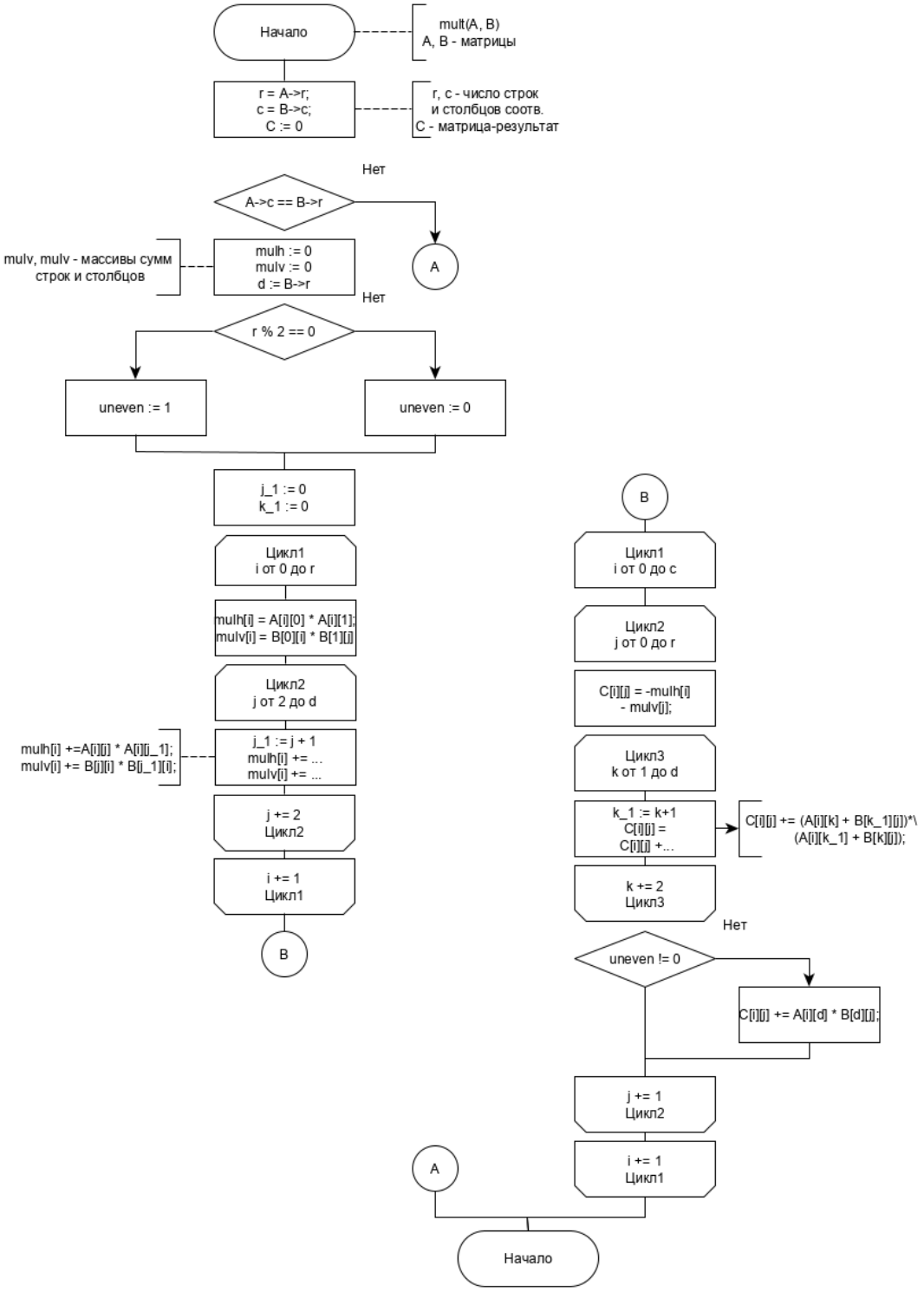


Рис. 3: Оптимизация Алгоритма Винограда

Для оптимизации алгоритма Винограда было внесено несколько изменений:

- Добавлено высчитывание констант.
- Предварительное вычисление сумм для матрицы было объединено в один цикл.
- Цикл по вычислению элементов в случае нечетной матрицы был объединен в основным циклом алгоритма.
- Шаг цикла увеличен в два раза.

- Пересчитаны индексы, замена 2*j-1 на j+1, которая высчитывается заранее.

2.3 Выводы по конструкторскому разделу

Таким образом, был определен стиль кода для реализации алгоритмов, а также разработаны схемы каждого из исследуемых алгоритмов.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна поддерживать два режима - пользовательский и экспериментальный.

В пользовательском режиме на вход подаются размеры целочисленного типа каждой из матриц, в консоль должны выводиться поданные матрицы и результат их произведения.

В экспериментальном режиме должно производиться тестирование времени работы каждого из алгоритмов на матрицах, размерностей $100 \times 100, \ldots, 1000 \times 1000$ и $101 \times 101, \ldots, 1001 \times 1001$ с шагом 100. Данные записываются в файл.

3.2 Средства реализации

Поскольку программа реализована в структурном стиле, из языков, поддерживающих его был выбран C++, так как на нем имеется наибольший опыт работы.

3.3 Листинг кода

Листинг 1: Стандартный алгоритм умножения матриц

```
mtx_t *standart(mtx_t *A, mtx_t *B)
{
    mtx_t *res = nullptr;
    int r = A->r;
    int c = B->c;

    if (check(A->c, B->r))
    {
        res = generate_mtx(r, c);

        for(int i = 0; i < r; i++)
            for(int j = 0; j < c; j++)
            {
                for(int k = 0; k < A->c; k++)
                    res->mtx[i][j] += A->mtx[i][k] * B->mtx[k][j];
            }
    }
    return res;
}
```

Листинг 2: Алгоритм Винограда

```
mtx_t *vinograd(mtx_t *A, mtx_t *B)
    size_t mtx_t *res = nullptr;
    int r = A->r;
    int c = B->c;

    if (check(A->c, B->r))
    {
        int *mulh = new int[r];

        for (int i=0; i<r; i++){
            mulh[i] = A->mtx[i][0] * A->mtx[i][1];

            for (int j=2; j<=B->r/2; j++){
                mulh[i] = mulh[i] + A->mtx[i][2*j-2] * A->mtx[i][2*j-1];
                }
        }

        int *mulv = new int[c];

        for (int i=0; i<c; i++){
            mulv[i] = B->mtx[0][i] * B->mtx[1][i];

            for (int j=2; j<=B->r/2; j++)
                mulv[i] = mulv[i] + B->mtx[2*j-2][i] * B->mtx[2*j-1][i];
        }


        res = generate_mtx(r, c);

        for (int i=0; i<r; i++)
            for (int j=0; j<c; j++){
                res->mtx[i][j] = -mulh[i] - mulv[j];

                for (int k=1; k<=B->r/2; k++)
                    res->mtx[i][j] = res->mtx[i][j] + \
                        (A->mtx[i][2*k-2]+B->mtx[2*k-1][j])*(A->mtx[i][2*k-1]+\
                        B->mtx[2*k-2][j]);
        }

        if (r%2 != 0)
            for (int i=0; i<r; i++)
                for (int j=0; j<c; j++)
                    res->mtx[i][j] = res->mtx[i][j] + A->mtx[i][B->r-1] *
                       B->mtx[B->r-1][j];

        delete []mulh;
        delete []mulv;
    }

    return res;
}
```

Листинг 3: Оптимизированный алгоритм Винограда

```
mtx_t *super_vinograd(mtx_t *A, mtx_t *B)
{
    mtx_t *res = nullptr;

    if (check(A->c, B->r))
    {
        int r = A->r;
        int c = B->c;
        int d = B->r-1;

        bool uneven = r % 2 != 0 ? 1 : 0;

        int *mulh = new int[r];
        int *mulv = new int[c];
        int j_1;


        for (int i=0; i<r; i++){

            mulh[i] = A->mtx[i][0] * A->mtx[i][1];
            mulv[i] = B->mtx[0][i] * B->mtx[1][i];

            for (int j=2; j<d; j+=2){

                j_1 = j+1;

                mulh[i] += A->mtx[i][j] * A->mtx[i][j_1];
                mulv[i] += B->mtx[j][i] * B->mtx[j_1][i];

            }
        }

        res = generate_mtx(r, c);
        int k_1;


        for (int i=0; i<r; i++)
            for (int j=0; j<c; j++){
                res->mtx[i][j] = -mulh[i] - mulv[j];


                for (int k=0; k<d; k+=2){
                    k_1 = k+1;
                    res->mtx[i][j] += (A->mtx[i][k] + B->mtx[k_1][j])*\
                    (A->mtx[i][k_1] + B->mtx[k][j]);
                }

                if (uneven)
                    res->mtx[i][j] += A->mtx[i][d] * B->mtx[d][j];
            }

        delete []mulh;
```

```
        delete []mulv;
    }

    return res;
}
```

3.4 Выводы по конструкторскому разделу

Были описаны требования ПО, приведен листинг кода с реализацией каждого из алгоритмов, а также был выбран язык программирования.

4 Экспериментальная часть

4.1 Примеры работы

Входные данные:

Input the 1st matrix size: **4**
Input the 2nd matrix size: **4**

Выходные данные:

Matrix A:
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
Matrix B:
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
C = A*B Standart:
0 6 12 18
0 6 12 18
0 6 12 18
0 6 12 18
C = A*B Winograd:
0 6 12 18
0 6 12 18
0 6 12 18
0 6 12 18
C = A*B Super_Winograd:
0 6 12 18
0 6 12 18
0 6 12 18
0 6 12 18

Входные данные:

---

1. User-mode;
2. Time testing-mode;
0. Exit;
Choose number: 5

Выходные данные:

---

incorrect input!

4.2 Постановка эксперимента

Исследуем работу трех алгоритмов и сравним время выполнения при обработке матриц размером $100 \times 100, \ldots, 1000 \times 1000$ и $101 \times 101, \ldots, 1001 \times 1001$ с шагом 100. При тестировании будем брать усредненное время из 5 тестов. Время будем замерять в секундах. С помощью библиотеки `ctime`

4.2.1 Тестирование времени работы функций

В таблицах 1 и 2 отображено время работы каждого алгоритма при обработке квадратных матриц четной и нечетной величин.

Матрица 1	Матрица 2	Стандартный,с	Виноград,с	Оптимизированный Виноград,с
100	100	0.0072	0.005	0.0034
200	200	0.0436	0.0354	0.0342
300	300	0.1634	0.1322	0.1166
400	400	0.3874	0.349	0.3
500	500	0.9068	0.6476	0.5932
600	600	2.4348	2.1198	1.0246
700	700	3.3512	3.284	1.6994
800	800	6.2290	3.4958	2.741
900	900	11.0358	6.9586	3.555
1000	1000	18.7464	6.4994	6.3508

Таблица 1: Тестирование времени на матрицах четного размера

Матрица 1	Матрица 2	Стандартный,с	Виноград,с	Оптимизированный Виноград,с
101	101	0.0054	0.0094	0.0044
201	201	0.0546	0.0368	0.0326
301	301	0.1632	0.1334	0.1234
401	401	0.364	0.3044	0.2896
501	501	0.7662	0.624	0.5898
601	601	1.341	1.084	1.0306
701	701	4.3538	2.1706	1.7386
801	801	7.1242	2.7058	2.7242
901	901	10.2698	3.6592	3.5614
1001	1001	14.7714	6.5754	6.2808

Таблица 2: Тестирование времени на матрицах нечетного размера

4.2.2 Память

Для работы с матрицей используется структура:

```
typedef struct mtx{
int **mtx;
int r;
int c;
}mtx_t;
```

Стандартный алгоритм

Для стандартного алгоритма умножения матриц используются указатели на обрабатываемые матрицы, внутри используется указатель на результрующую матрицу, а также переменные для циклов

$Memory = sizeof(mtx\_t*) * 3 + 2 * sizeof(int) + n * sizeof(int*) + n^2 * size(int)$
Для матрицы 100×100 потребуется: 4856 Б

Алгоритм Винограда

Для алгоритма Винограда дополнительно используются два массива для хранения сумм строк и столбцов, поэтому данный алгоритм требует больше памяти:

$Memory = sizeof(mtx\_t*) * 3 + 2 * sizeof(int) + n * sizeof(int*) + n^2 * size(int) + 2 * n * sizeof(int) + 2 * sizeof(int*)$
Для матрицы 100×100 потребуется: 5672 Б

Оптимизированный алгоритм Винограда

Оптимизированный алгоритм требует большей всех памяти, так как добавляются локальные переменные, в которых хранятся заранее вычисленные значения, поэтому

$Memory = sizeof(mtx\_t*) * 3 + 2 * sizeof(int) + n * sizeof(int*) + n^2 * size(int) + 4 * n * sizeof(int) + 2 * sizeof(int*) + sizeof(bool)$
Для матрицы 100×100 потребуется: 5681 Б

4.2.3 Трудоемкость алгоритмов

Пусть стоимость базовых операций $+, -, *, /, \%, =, ==, !=, <, >, [], ++, +=, --$ равна 1.

- R - количество строк в матрице A;
- C - количество столбцов в матрице B;
- N - количество столбцов, строк в матрицах A, B соответственно.
- Стоимость цикла 2 + N(2 + T), где T - тело, внутри цикла.

Стандартный алгоритм

Общая трудоемкость алгоритма: $O(N^3)$
Тогда трудоемкость стандартного алгоритма это:

$$2 + R(2 + 2 + C(2 + 2 + N(2 + 8 + 1 + 1 + 1))) = 2 + 4R + 4RC + 13RCN \quad (4)$$

Алгоритм Винограда

Трудоемкость алгоритма Винограда: Так как таких циклов в программе два - увеличим результат в два раза, получим:

$$f_1 = 2 * (2 + R(2 + 5 + 1 + 1 + (\frac{N}{2} - 2)(2 + 6 + 1 + 1 + 3 + 2))) = 15NR - 42R + 4 \quad (5)$$

$$f_2 = 2+R(2+2+C(2+4+2+1+(\frac{N}{2}-1)(2+10+4+4+3+1)) = 12NCR+15CR+4R+2 \quad (6)$$

$$f_{all} = f_1 + f_2 = 12NRC + 15CR - 38R + 6 \quad (7)$$

Если матрица нечетная:

$$f_3 = 2 + R(2 + 2 + C(2 + 8 + 1 + 1 + 1 + 2)) = 15CR + 4R + 2 \quad (8)$$

$$f_{all} = f_1 + f_2 + f_3 = 12NCR - 34R + 8 \quad (9)$$

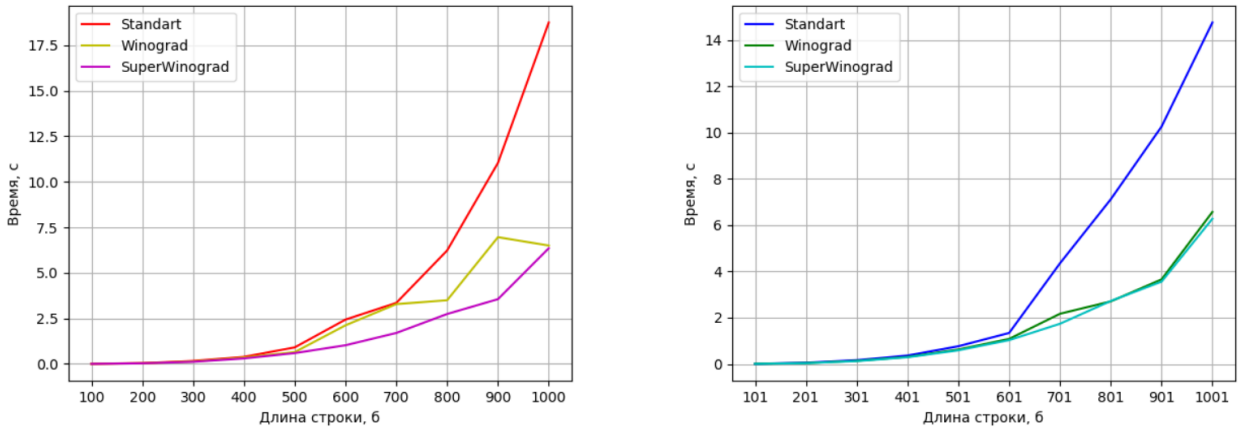
Оптимизированный алгоритм Винограда

$$f_1 = 2+R(2+2+4+2+1+C(2+2+\frac{N}{2}(2+1+10+1+2)+1+6+1+1)) = 11R+13CR+8NCR+2 \quad (10)$$

$$f_2 = 2 + r(2 + 2 + 5 + 1 + 1 + 2 + 5 + 1 + 1 + \frac{N}{2}(2 + 10 + 2 + 2 + 1 + 1)) = 9NR + 20R + 2 \quad (11)$$

$$f_{all} = f_1 + f_2 = 4 + 31R + 13CR + 8NCR \quad (12)$$

4.3 Сравнительный анализ на материале экспериментальных данных



а) б)

Рис. 4: Сравнение работы алгоритмов

на рис. 4 отображена работа трех алгоритмов при матрицах четной и нечетной размерности.

4.4 Выводы по экспериментальной части

На основе полученных в ходе эксперимента данных можно сделать ряд заключений:

- При матрицах небольшого размера$(100 \times 100, \ldots, 400 \times 400)$ все алгоритмы показывают примерно одинаковый результат: стандартный алгоритм отработал за 0.3874с, алгоритм Винограда 0.349с оптимизированный алгоритм - 0.3.
- При работе с матрицами больших размеров алгоритмы Винограда и его оптимизация работают быстрее стандартного для 1000 элементов первый алгоритм отработал за 18.7464, второй за 6.4994, 6.3508с.
- С увеличением данных растет разница в скорости алгоритмов - для матриц 700×700 алгоритм Винограда работает почти в два раза быстрее стандартного, при этих значениях почти в три раза.

Заключение

Таким образом, в ходе лабораторной работы были реализованы алгоритмы умножения матриц, написана оптимизация одного из них. Работа каждого алгоритма была протестирована на четных и нечетных матрицах, в ходе эксперимента было взято усредненное время, для получения наиболее точного результата. На основе полученных результатов был сделан вывод, что для матриц размером до $\approx 400 \times 400$ алгоритмы работают с одинаковой скоростью,стандартный алгоритм отработал за 0.3874с, алгоритм Винограда 0.349с оптимизированный алгоритм - 0.3. однако стандартному алгоритму требуется меньше памяти, поэтому он является оптимальным для подобного объема данных. Для матриц большего размера оптимальными являются алгоритм Винограда и его модификация, которые показали результат в три раза лучше стандартного при обработке матрицы 1000×1000. Для умножения матриц размера 1000 отработал за 18.7464, второй за 6.4994, 6.3508с. При обработке нечетных значений можно сделать вывод, что при подобной реализации алгоритм Винограда и его оптимизация работают примерно с одинаковой скоростью алгоритм Винограда отработал за 6.5754с, оптимизированный за 6.2808с.

Список литературы

[1] А.Н. Канатников, А.П. Крищенко, "Линейная алгебра и функции нескольких переменных"

[2] Винсковская Л.А., Маркушина А.А. "Способы реализации экономико-математических моделей. Современное развитие России в условиях новой цифровой экономики материалы II Международной научнопрактической конференции"

[3] В. Н. Задорожный, В. Ф. Зальмеж, А.Ю. Трифонов, А. В. Шаповалов, "Высшая математика для технических университетов. Линейная алгебра"

[4] Е. Е. Тыртышников "Матричный анализ и линейная алгебра"

[5] Jeffrey J. McConnell, "Analysis of Algorithms: An Active Learning Approach"

[6] Левитин А. В. Глава 4. Метод декомпозиции: Умножение больших целых чисел и алгоритм умножения матриц Штрассена