ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

Потоки: Нахождение обратной квадратной матрицы.

Пояснительная записка

Исполнитель:
студент группы БПИ199
Вахитова Д. Р.
«17» ноября 2020 г.

Москва 2020

Оглавление:

1. Текст задания.

Найти обратную матрицу для матрицы A. Входные данные: целое положительное число n, произвольная матрица A размерности n x n. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

2. Применяемые расчетные методы.
Понятие обратной матрицы вводится лишь для квадратных матриц, определитель которых отличен от нуля, то есть для невырожденных квадратных матриц.
Матрица A^{-1} называется обратной для матрицы А, отпределитель которой отличен от 0, если справедливы равенства: $A*A^{-1}=E$, где Е – единичная матрица.

Существуют альтернативные методы нахождения обратной матрицы, например, **метод Гаусса - Жордана.**

Суть метода Гаусса-Жордана заключается в том, что если с единичной матрицей *E* провести элементарные преобразованиия, которыми невырожденная квадратная матрица *A* приводится к *E*, то получится обратная матрица A^{-1}.

Опишем алгоритм приведения матрицы *A* порядка *n* на *n*, определитель которой не равен нулю, к единичной матрице методом Гаусса - Жордана. После описания алгоритма разберем пример, чтобы все стало понятно.

Сначала преобразуем матрицу так, чтобы элемент a_{11} стал равен единице, а все остальные элементы первого столбца стали нулевыми.

Если $a_{11}=0$, то на место первой строки ставится *k-ая* строка (*k>1*), в которой $a_{k1}\neq 0$, а на место *k-ой* строки ставится первая. (Строка с $a_{k1}\neq 0$ обязательно существует, в противном случае матрица *A* – вырожденная). После перестановки строк получили «новую» матрицу *A*, у которой $a_{11}\neq 0$.

Теперь умножаем каждый элемент первой строки на $\frac{1}{a_{11}}$. Так приходим к «новой» матрице *A*, у которой $a_{11}=1$. Далее к элементам второй строки прибавляем соответствующие элементы первой строки, умноженные на $-a_{21}$. К элементам третьей строки – соответствующие элементы первой строки, умноженные на $-a_{31}$. И продолжаем такой процесс до *n-ой* строки включительно. Так все элементы первого столбца матрицы *A*, начиная со второго, станут нулевыми.

С первым столбцом разобрались, переходим ко второму.

Преобразуем матрицу *A* так, чтобы элемент a_{22} стал равен единице, а все остальные элементы второго столбца, начиная с a_{32}, стали нулевыми.

Если $a_{22}=0$, то на место второй строки ставится *k-ая* строка (*k*>2), в которой $a_{k2}\neq 0$, а на место *k-ой* строки ставится вторая. Так получаем преобразованную матрицу *A*, у которой $a_{22}\neq 0$. Умножаем все элементы второй строки на $\frac{1}{a_{22}}$. После этого к элементам третьей строки прибавляем соответствующие элементы второй строки, умноженные на $-a_{32}$. К элементам четвертой строки – соответствующие элементы второй строки, умноженные на $-a_{42}$. И продолжаем такой процесс до *n-ой* строки включительно. Так все элементы второго столбца матрицы *A*, начиная с третьего, станут нулевыми, а a_{22} будет равен единице.

Со вторым столбцом закончили, переходим к третьему и проводим аналогичные преобразования.

Так продолжаем процесс, пока все элементы главной диагонали матрицы *A* не станут равными единице, а все элементы ниже главной диагонали не станут равными нулю.

С этого момента начинаем обратный ход метода Гаусса-Жордана. Теперь преобразуем матрицу *A* так, чтобы все элементы *n-ого* столбца, кроме a_{nn}, стали нулевыми. Для этого к элементам *(n-1)-ой* строки прибавляем соответствующие элементы *n-ой* строки, умноженные на $-a_{n-1\,n}$. К элементам *(n-2)-ой* строки – соответствующие элементы *n-ой* строки, умноженные на $-a_{n-2\,n}$. И продолжаем такой процесс до первой строки включительно. Так все элементы *n-ого* столбца матрицы *A* (кроме a_{nn}), станут нулевыми.

С последним столбцом разобрались, переходим к *(n-1)-ому*.

Преобразуем матрицу *A* так, чтобы все элементы *(n-1)-ого* столбца до $a_{n-1\,n-1}$, стали нулевыми. Для этого к элементам *(n-2)-ой* строки прибавляем соответствующие элементы *(n-1)-ой* строки, умноженные на $-a_{n-2\,n-1}$. К элементам *(n-3)-ой* строки – соответствующие элементы *(n-1)-ой* строки, умноженные на $-a_{n-3\,n-1}$. И продолжаем такой процесс до первой строки включительно. Так все элементы *(n-1)-ого* столбца матрицы *A* (кроме $a_{n-1\,n-1}$), станут нулевыми.

Действуя дальше схожим образом, мы получим единичную матрицу.

В программе также используются потоки из стандартной библиотеки с++ std::thread.

3. Список используемых источников.

[1] Инструкция по составлению пояснительной записки [Электронный ресурс]. //URL: http://softcraft.ru/edu/comparch/tasks/mp01/ (Дата обращения: 28.10.2020, режим доступа: свободный)

[2] Finding inverse of a matrix using Gauss – Jordan Method [Электронный ресурс]. //URL: https://www.geeksforgeeks.org/finding-inverse-of-a-matrix-using-gauss-jordan-method/ (Дата обращения: 28.10.2020, режим доступа: свободный)

[3] Нахождение обратной матрицы методом Гаусса [Электронный ресурс]. //URL: http://www.cleverstudents.ru/matrix/finding_the_inverse_matrix.html (Дата обращения: 28.10.2020, режим доступа: свободный)

[4] std::thread [Электронный ресурс]. //URL: https://en.cppreference.com/w/cpp/thread/thread (Дата обращения: 28.10.2020, режим доступа: свободный)

4. Описание работы программы.

Инклуды стандартной библиотеки.

```
#include <iostream>
#include <fstream>
#include <thread>
#include <sstream>
#include <vector>
```

Программа работает с командной строкой. В качестве единственного параметра передается путь к файлу в котором в стандартном виде хранится число, отвечающее за количество строк и столбцов, потоков, плюс матрица записанная по столбцам:

```
test – Блокнот
Файл  Правка  Формат  Вид  Справка
3 2
1 4 5
2 3 4
3 4 5
```

Главный метод программы. Матрицы - вектор векторов. Ввод осуществляется через файл.

```
int main(int argsNumber, char** args) {
    // Обработка аргументов командной строки.
    if (argsNumber != 2) {
        std::cout << "Wrong Console Data" << std::endl;
        std::exit(EXIT_FAILURE);
    }
    // Для оптимизации используется работа с файлами.
    int N, t;
    std::ifstream in{ args[1] };
    in >> N >> t;
    std::vector<std::vector<float>> matrix(N);
    std::vector<float> temp(N);
    // Заполняем матрицу.
    for (auto i = 0; i < N; ++i) {
        for (auto j = 0; j < N; ++j) {
            in >> temp[j];
        }
        matrix[i] = temp;
    }

    // Находим обртаную.
    InverseOfMatrix(matrix, N);

    // Начинаем работать с потоками.
    std::vector<std::thread> threads;
    for (int i = 0; i < t; ++i) {
        threads.emplace_back(threadFunc, &matrix);
    }
    // Объединяем потоки.
    for (int i = 0; i < t; ++i) threads[i].join();

    // Печатаем результат.
    printf("\n=== Inversed Matrix ===\n");
    PrintInverse(matrix, N, N * 2);
}
```

Метод, который отвечает за нахождение обратной матрицы. Смотреть подробнее в пункте 2.

```
// Нахождение обратной матрицы.
void InverseOfMatrix(vector<vector<float>>& matrix, int order) {
    float temp_;
    // Печать первоначальной матрицы.
    printf("=== Matrix ===\n");
    PrintMatrix(matrix, order, order);

    for (int i = 0; i < order; ++i) matrix[i].resize(2 * order);
    for (int i = 0; i < order; ++i) {

        for (int j = 0; j < 2 * order; ++j) {
            if (j == (i + order))
                matrix[i][j] = 1;
        }
    }
    for (int i = order - 1; i > 0; --i) {
        if (matrix[i - 1][0] < matrix[i][0]) {
            vector<float> temp = matrix[i];
            matrix[i] = matrix[i - 1];
            matrix[i - 1] = temp;
        }
    }

    printf("\n=== Augmented Matrix ===\n");
    PrintMatrix(matrix, order, order * 2);
    // Тут в происходит обратный ход Гаусса, я бы сказала это неудобно параллелить.
    for (int i = 0; i < order; i++) {
        for (int j = 0; j < order; j++) {
            if (j != i) {
                temp_ = matrix[j][i] / matrix[i][i];
                for (int k = 0; k < 2 * order; k++) {
                    matrix[j][k] -= matrix[i][k] * temp_;
                }
            }
        }
    }
```

В методе Гаусса параллельность применить тяжело. Используется для того, чтобы превратить диагональную матрицу в единичную. Разные потоки отвечают за разные строки диагональной матрицы.

```
void threadFunc(vector<vector<float>>* A) {
    vector<vector<float>>& matrix = *A;
    for (int i = 0; i < matrix.size(); ++i) {
        float temp_ = matrix[i][i];
        for (int j = 0; j < matrix.size() * 2; ++j) {
            matrix[i][j] = matrix[i][j] / temp_;
        }
    }
}
```

Просто метод для вывода матрицы.

```
// Вывод матрицы.
void PrintMatrix(vector<vector<float>>& ar, int n, int m)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) cout << ar[i][j] << "  ";
        printf("\n");
    }
    return;
}
```

Программа выводит на консоль исходную, дополненную и обратную матрицы с точностью до 3 знаков после запятой.

5. Тестирование программы.

Пример 1: На матрице 5 на 5. Результат работы программы совпадает с

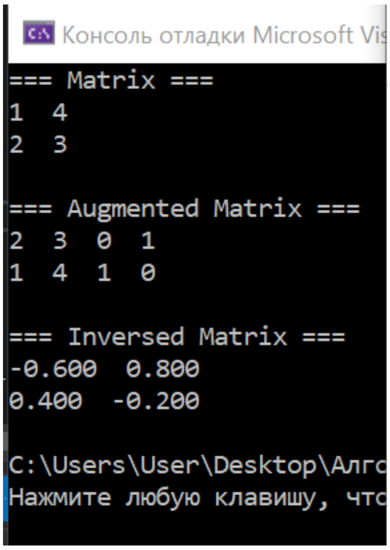
```
=== Matrix ===
1  2  3  4  5
6  3  4  5  6
3  4  5  6  7
7  5  6  1  3
3  7  5  4  4

=== Augmented Matrix ===
7  5  6  1  3  0  0  0  1  0
1  2  3  4  5  1  0  0  0  0
6  3  4  5  6  0  1  0  0  0
3  4  5  6  7  0  0  1  0  0
3  7  5  4  4  0  0  0  0  1

=== Inversed Matrix ===
-0.125  0.250  -0.125  -0.000  -0.000
1.317  0.365  -1.529  -0.077  0.538
-2.077  -0.846  2.462  0.231  -0.615
-2.798  -0.404  2.663  -0.231  -0.385
3.183  0.635  -2.971  0.077  0.462
```

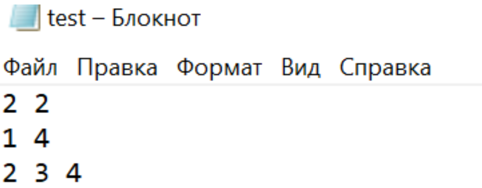
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 6 & 1 & 3 \\ 3 & 7 & 5 & 4 & 4 \end{pmatrix}^{(-1)} = \begin{pmatrix} -0{,}125 & 0{,}25 & -0{,}125 & 0 & 0 \\ 1{,}32 & 0{,}365 & -1{,}53 & -0{,}0769 & 0{,}538 \\ -2{,}08 & -0{,}846 & 2{,}46 & 0{,}231 & -0{,}615 \\ -2{,}80 & -0{,}404 & 2{,}66 & -0{,}231 & -0{,}385 \\ 3{,}18 & 0{,}635 & -2{,}97 & 0{,}0769 & 0{,}462 \end{pmatrix}$$

Пример 2: Крайний случай: матрица из одно элемента. По факту, нахождение обратного элемента.

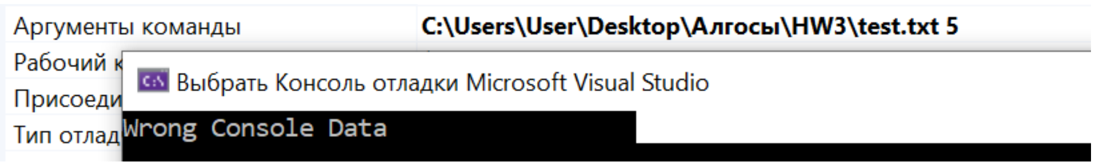
```
=== Matrix ===
4

=== Augmented Matrix ===
4  1

=== Inversed Matrix ===
0.250
```

Пример 3: Ввод побитой матрицы. По строкам обрезаются лишние символы.





Пример 4: При некорректных аргументах командной строки выводится сообщение об этом.



Приложение №1.

Код программы:

```
/**
 * Вариант задания 4.
 * Нахождение обратной матрицы для квадратной матрицы размера nxn.
 * Выполнила студентка группы БПИ 199 Вахитова Диана.
 */
#include <iostream>
#include <fstream>
#include <thread>
#include <sstream>
#include <vector>

using namespace std;

// Вывод матрицы.
void PrintMatrix(vector<vector<float>>& ar, int n, int m)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) cout << ar[i][j] << "  ";
        printf("\n");
    }
    return;
}

// Вывод обратной матрицы.
void PrintInverse(vector<vector<float>>& ar, int n, int m)
{
    for (int i = 0; i < n; ++i) {
        for (int j = n; j < m; ++j) printf("%.3f  ", ar[i][j]);
        printf("\n");
    }
    return;
}

// Нахождение обратной матрицы.
void InverseOfMatrix(vector<vector<float>>& matrix, int order) {
    float temp_;
    // Печать первоначальной матрицы.
    printf("=== Matrix ===\n");
    PrintMatrix(matrix, order, order);

    for (int i = 0; i < order; ++i) matrix[i].resize(2 * order);
    for (int i = 0; i < order; ++i) {

        for (int j = 0; j < 2 * order; ++j) {
            if (j == (i + order))
                matrix[i][j] = 1;
        }
    }
    for (int i = order - 1; i > 0; --i) {
        if (matrix[i - 1][0] < matrix[i][0]) {
            vector<float> temp = matrix[i];
            matrix[i] = matrix[i - 1];
            matrix[i - 1] = temp;
        }
    }

    printf("\n=== Augmented Matrix ===\n");
    PrintMatrix(matrix, order, order * 2);
    // Тут в происходит обратный ход Гаусса, я бы сказала это неудобно параллелить.
    for (int i = 0; i < order; i++) {
        for (int j = 0; j < order; j++) {
            if (j != i) {
                temp_ = matrix[j][i] / matrix[i][i];
                for (int k = 0; k < 2 * order; k++) {
                    matrix[j][k] -= matrix[i][k] * temp_;
                }
            }
```

```
        }
    }
    return;
}

// Метод делает из диагональной маттрицы единичную. Невероятная польза, это происходит в t раз быстрее.
void threadFunc(vector<vector<float>>* A) {
    vector<vector<float>>& matrix = *A;
    for (int i = 0; i < matrix.size(); ++i) {
        float temp_ = matrix[i][i];
        for (int j = 0; j < matrix.size() * 2; ++j) {
            matrix[i][j] = matrix[i][j] / temp_;
        }
    }
}

int main(int argsNumber, char** args) {
    // Обработка аргументов командной строки.
    if (argsNumber != 2) {
        std::cout << "Wrong Console Data" << std::endl;
        std::exit(EXIT_FAILURE);
    }
    // Для оптимизации используется работа с файлами.
    int N, t;
    std::ifstream in{ args[1] };
    in >> N >> t;
    std::vector<std::vector<float>> matrix(N);
    std::vector<float> temp(N);
    // Заполняем матрицу.
    for (auto i = 0; i < N; ++i) {
        for (auto j = 0; j < N; ++j) {
            in >> temp[j];
        }
        matrix[i] = temp;
    }

    // Находим обртаную.
    InverseOfMatrix(matrix, N);

    // Начинаем работать с потоками.
    std::vector<std::thread> threads;
    for (int i = 0; i < t; ++i) {
        threads.emplace_back(threadFunc, &matrix);
    }
    // Объединяем потоки.
    for (int i = 0; i < t; ++i) threads[i].join();

    // Печатаем результат.
    printf("\n=== Inversed Matrix ===\n");
    PrintInverse(matrix, N, N * 2);

 }
```