МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5d

по дисциплине: Основы программирования
тема: «Работа с многомерными массивами»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2022 г.

Цель работы: получение навыков работы с многомерными массивами.

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Решения задач:
  - Текст задания.
  - Исходный код (в том числе и тестов).
  - Задания со звездочкой не являются обязательными, но их решение требуется для получения максимального балла.
- Ссылка на открытый репозиторий с решением.
- Скриншот с историей коммитов.

Ссылка на репозиторий: https://github.com/ChuvilkoDEV/lab-5d

Содержимое файла matrix.h:

```c
#ifndef INC_5D_MATRIX_H
#define INC_5D_MATRIX_H

#include <stdio.h>
#include <malloc.h>
#include <stdbool.h>
#include <assert.h>
#include <math.h>
#include "../../algorithms/algorithms.h"

#define FLOAT_EPS 0.00001

typedef struct matrix {
  int **values; // элементы матрицы
  int nRows; // количество рядов
  int nCols; // количество столбцов
} matrix;

typedef struct position {
  int rowIndex;
  int colIndex;
} position;

matrix getMemMatrix(int nRows, int nCols);

matrix *getMemArrayOfMatrices(int nMatrices, int nRows, int nCols);

void freeMemMatrix(matrix m);

void freeMemMatrices(matrix *ms, int nMatrices);

void inputMatrix(matrix m);

void inputMatrices(matrix *ms, int nMatrices);

void outputMatrix(matrix m);

void outputMatrices(matrix *ms, int nMatrices);

void swapRows(matrix *m, int i1, int i2);

void swapColumns(matrix *m, int j1, int j2);

void insertionSortRowsMatrixByRowCriteria(
  matrix *m, int (*criteria)(int *, int));

void insertionSortColsMatrixByColCriteria(
  matrix *m, int (*criteria)(int *, int));
```

```
bool isSquareMatrix(matrix m);

bool twoMatricesEqual(matrix m1, matrix m2);

bool isEMatrix(matrix m);

bool isSymmetricMatrix(matrix m);

void transposeSquareMatrix(matrix *m);

position getMinValuePos(matrix m);

position getMaxValuePos(matrix m);

matrix createMatrixFromArray(const int *a, int nRows, int nCols);

matrix *createArrayOfMatrixFromArray(
  const int *values, int nMatrices, int nRows, int nCols);

void exchangeMaxAndMinRow(matrix *m);

void sortRowsByMaxElement(matrix *m);

void sortColsByMinElement(matrix *m);

matrix mulMatrices(matrix m1, matrix m2);

void getSquareOfMatrixIfSymmetric(matrix *m);

void transposeIfMatrixHasNotEqualSumOfRows(matrix m);

bool isMutuallyInverseMatrices(matrix m1, matrix m2);

long long findSumOfMaxesOfPseudoDiagonal(matrix m);

int getMinInArea(matrix m);

float getDistance(int *a, int n);

void sortByDistances(matrix *m);

int countNUnique(long long *a, int n);

int getNSpecialElement(matrix m);

position getLeftMin(matrix m);

void swapPenultimateRow(matrix *m, int n);

bool isNonDescendingSorted(int const *a, int n);

bool hasAllNonDescendingRows(matrix m);

int countNonDescendingRowsMatrices(matrix *ms, int nMatrix);

int countValues(const int *a, int n, int value);

int countZeroRows(matrix m);

void printMatrixWithMaxZeroRows(matrix *ms, int nMatrix);

void minimOfNorms(matrix *ms, int nMatrix);

#endif //INC_5D_MATRIX_H
```

Содержимое файла matrix.c:

```c
#include "matrix.h"

// Реализация библиотеки

// Задание 2 (a)
// Размещает в динамической памяти матрицу размером nRows на nCols.
// Возвращает матрицу.
matrix getMemMatrix(int nRows, int nCols) {
  int **values = (int **) malloc(sizeof(int *) * nRows);
  for (int i = 0; i < nRows; i++)
    values[i] = (int *) malloc(sizeof(int) * nCols);
  return (matrix) {values, nRows, nCols};
}

// Задание 2 (b)
// Размещает в динамической памяти массив из nMatrices матриц размером
// nRows на nCols. Возвращает указатель на нулевую матрицу.
matrix *getMemArrayOfMatrices(int nMatrices,
                              int nRows, int nCols) {
  matrix *ms = (matrix *) malloc(sizeof(matrix) * nMatrices);
  for (int i = 0; i < nMatrices; i++)
    ms[i] = getMemMatrix(nRows, nCols);
  return ms;
}

// Задание 2 (c)
// Освобождает память, выделенную под хранение матрицы m.
void freeMemMatrix(matrix m) {
  for (int i = 0; i < m.nRows; i++)
    free(m.values[i]);
  free(m.values);
}

// Задание 2 (d)
// Освобождает память, выделенную под хранение массива ms из nMatrices матриц.
void freeMemMatrices(matrix *ms, int nMatrices) {
  for (int i = 0; i < nMatrices; i++)
    freeMemMatrix(ms[i]);
  free(ms);
}

// Задание 3 (a)
// Ввод матрицы m.
void inputMatrix(matrix m) {
  for (int i = 0; i < m.nRows; i++)
    for (int j = 0; j < m.nCols; j++)
      scanf("%d ", &m.values[i][j]);
}

// Задание 3 (b)
// Ввод массива из nMatrices матриц, хранящейся по адресу ms.
void inputMatrices(matrix *ms, int nMatrices) {
  for (int i = 0; i < nMatrices; i++)
    inputMatrix(ms[i]);
}
```

```c
// Задание 3 (c)
// Вывод матрицы m
void outputMatrix(matrix m) {
  if (m.nCols == 0 || m.nRows == 0) {
    printf("Empty Matrix");
    return;
  }
  printf("[");
  for (int i = 0; i < m.nRows; i++) {
    printf("[");
    for (int j = 0; j < m.nCols; j++)
      printf("%d, ", m.values[i][j]);
    printf("\b\b], ");
  }
  printf("\b\b]");
}

// Задание 3 (d)
// Вывод массива из nMatrices матриц, хранящейся по адресу ms.
void outputMatrices(matrix *ms, int nMatrices) {
  printf("[");
  for (int i = 0; i < nMatrices; i++) {
    outputMatrix(ms[i]);
    printf(", ");
  }
  printf("\b\b]");
}

// Задание 4 (a)
// Обмен строк с порядковыми номерами i1 и i2 в матрице m.
void swapRows(matrix *m, int i1, int i2) {
  int *t = m->values[i1];
  m->values[i1] = m->values[i2];
  m->values[i2] = t;
}
// Задание 4 (b)
// Обмен колонок с порядковыми номерами j1 и j2 в матрице m
void swapColumns(matrix *m, int j1, int j2) {
  for (int i = 0; i < m->nRows; i++)
    swap(&m->values[i][j1], &m->values[i][j2]);
}

// Задание 5 (a)
// выполняет сортировку вставками строк матрицы m по неубыванию значения
// функции criteria применяемой для строк
void insertionSortRowsMatrixByRowCriteria(
  matrix *m, int (*criteria)(int *, int)) {
  int rowsCriteria[m->nRows];
  for (int i = 0; i < m->nRows; i++)
    rowsCriteria[i] = criteria(m->values[i], m->nCols);

  for (int i = 1; i < m->nRows; i++)
    for (int j = i; j > 1; j--)
      if (rowsCriteria[j] < rowsCriteria[j - 1]) {
        swap(&rowsCriteria[j], &rowsCriteria[j - 1]);
        swapRows(m, j, j - 1);
      } else break;
}
```

```
// Задание 5 (b)
// выполняет сортировку выбором столбцов матрицы m по неубыванию значения
// функции criteria применяемой для столбцов
void insertionSortColsMatrixByColCriteria(matrix *m,
                                          int (*criteria)(int *, int)) {
  int colsCriteria[m->nCols];
  for (int i = 0; i < m->nCols; i++) {
    int a[m->nRows];
    for (int j = 0; j < m->nRows; j++)
      a[j] = m->values[j][i];
    colsCriteria[i] = criteria(a, m->nRows);
  }

  for (int i = 1; i < m->nCols; i++)
    for (int j = i; j > 1; j--)
      if (colsCriteria[j] > colsCriteria[j - 1]) {
        swap(&colsCriteria[j], &colsCriteria[j - 1]);
        swapColumns(m, j, j - 1);
      } else break;
}

// Задание 6 (a)
// возвращает значение 'истина', если матрица m является квадратной, ложь –
// в противном случае
bool isSquareMatrix(matrix m) {
  return m.nRows == m.nCols;
}

// Задание 6 (b)
// возвращает значение 'истина', если матрицы m1 и m2 равны, ложь –
// в противном случае.
bool twoMatricesEqual(matrix m1, matrix m2) {
  if (m1.nCols != m2.nCols || m1.nRows != m2.nRows)
    return false;
  for (int i = 0; i < m1.nRows; i++)
    for (int j = 0; j < m1.nCols; j++)
      if (m1.values[i][j] != m2.values[i][j])
        return false;
  return true;
}

// Задание 6 (c)
// возвращает значение 'истина', если матрица m является единичной,
// ложь – в противном случае.
bool isEMatrix(matrix m) {
  int n = min(m.nCols, m.nRows);
  for (int i = 0; i < n; i++)
    if (m.values[i][i] != 1)
      return false;
  return true;
}

// Задание 6 (e)
// возвращает значение 'истина', если матрица m является симметричной,
// ложь – в противном случае.
bool isSymmetricMatrix(matrix m) {
  if (!isSquareMatrix(m))
    return false;
  for (int i = 0; i < m.nRows; i++)
    for (int j = i + 1; j < m.nCols; j++)
      if (m.values[i][j] != m.values[j][i])
        return false;
  return true;
}
```

```
// Задание 7 (a)
// транспонирует квадратную матрицу m.
void transposeSquareMatrix(matrix *m) {
  if (!isSquareMatrix(*m))
    return;
  for (int i = 0; i < m->nRows; i++)
    for (int j = i + 1; j < m->nCols; j++)
      swap(&m->values[i][j], &m->values[j][i]);
}

// Задание 8 (a)
// возвращает позицию минимального элемента матрицы m.
position getMinValuePos(matrix m) {
  position min = {1, 1};
  for (int i = 0; i < m.nRows; i++)
    for (int j = 0; j < m.nCols; j++)
      if (m.values[min.rowIndex][min.colIndex] > m.values[i][j])
        min = (position) {i, j};
  return min;
}

// Задание 8 (b)
// возвращает позицию максимального элемента матрицы m.
position getMaxValuePos(matrix m) {
  position max = {1, 1};
  for (int i = 0; i < m.nRows; i++)
    for (int j = 0; j < m.nCols; j++)
      if (m.values[max.rowIndex][max.colIndex] < m.values[i][j])
        max = (position) {i, j};
  return max;
}

// Задание 9 (a)
//возвращает матрицу размера nRows на nCols, построенную из элементов массива a
matrix createMatrixFromArray(const int *a, int nRows, int nCols) {
  matrix m = getMemMatrix(nRows, nCols);
  int k = 0;
  for (int i = 0; i < nRows; i++)
    for (int j = 0; j < nCols; j++)
      m.values[i][j] = a[k++];
  return m;
}

// Задание 9 (b)
// возвращает указатель на нулевую матрицу массива из nMatrices матриц,
// размещенных в динамической памяти, построенных из элементов массива a
matrix *createArrayOfMatrixFromArray(const int *values, int nMatrices,
                                     int nRows, int nCols) {
  matrix *ms = getMemArrayOfMatrices(nMatrices, nRows, nCols);
  int l = 0;
  for (int k = 0; k < nMatrices; k++)
    for (int i = 0; i < nRows; i++)
      for (int j = 0; j < nCols; j++)
        ms[k].values[i][j] = values[l++];
  return ms;
}
```

```
// Решение задач:


// Задание 1

// Меняет местами строки, в которых находятся минимальный и максимальный
// элементы
void exchangeMaxAndMinRow(matrix *m) {
  position min = getMinValuePos(*m);
  position max = getMaxValuePos(*m);

  swapRows(m, min.rowIndex, max.rowIndex);
}

// Задание 2

// Сортирует строки матрицы по неубыванию наибольших элементов строк
void sortRowsByMaxElement(matrix *m) {
  insertionSortRowsMatrixByRowCriteria(m, (int (*)(int *, int)) getMax);
}

// Задание 3

// Упорядочить столбцы матрицы по неубыванию минимальных элементов столбцов:
void sortColsByMinElement(matrix *m) {
  insertionSortColsMatrixByColCriteria(m, (int (*)(int *, int)) getMin);
}

// Задание 4

// Возвращает матрицу, которая является результатом произведения матрицы m1 и
m2
matrix mulMatrices(matrix const m1, matrix const m2) {
  assert(m1.nCols == m2.nRows);
  matrix m3 = getMemMatrix(m1.nRows, m2.nCols);

  for (int i = 0; i < m1.nRows; i++)
    for (int j = 0; j < m2.nCols; j++) {
      int s = 0;
      for (int k = 0; k < m1.nCols; k++)
        s += m1.values[i][k] * m2.values[k][j];
      m3.values[i][j] = s;
    }
  return m3;
}

// Возвращает матрицу, которая является квадратом квадратной матрицы
void getSquareOfMatrixIfSymmetric(matrix *m) {
  if (!isSymmetricMatrix(*m))
    return;
  matrix res = mulMatrices(*m, *m);
  *m = res;
  freeMemMatrix(res);
}

// Задание 5

// Если среди сумм элементов строк матрицы нет равных, то транспонирует
матрицу.
void transposeIfMatrixHasNotEqualSumOfRows(matrix m) {
  long long sumArray[m.nRows];
  for (int i = 0; i < m.nRows; i++)
    sumArray[i] = getSum(m.values[i], m.nCols);
  if (!isUnique(sumArray, m.nRows))
    return;
  transposeSquareMatrix(&m);
}
```

```
// Задание 6

// Возвращает (true), если матрицы m1 и m2 являются взаимообратными,
// (false) - в ином случае.
bool isMutuallyInverseMatrices(matrix m1, matrix m2) {
  matrix res = mulMatrices(m1, m2);
  if (isEMatrix(res)) {
    freeMemMatrix(res);
    return true;
  }
  return false;
}

// Задание 7

// Возвращает сумму максимальных элементов псевдодиагонали.
long long findSumOfMaxesOfPseudoDiagonal(matrix m) {
  long long s = 0;
  for (int i = 1; i < m.nRows; i++) {
    int localMax = m.values[i][0];
    for (int j = 1; j < min(m.nCols, m.nRows) - i; j++)
      localMax = max(localMax, m.values[i + j][j]);
    s += localMax;
  }
  for (int i = 1; i < m.nCols; i++) {
    int localMax = m.values[0][i];
    for (int j = 1; j < min(m.nCols, m.nRows) - i; j++)
      localMax = max(localMax, m.values[j][i + j]);
    s += localMax;
  }
  return s;
}

// Задание 8

// Возвращает минимальный элемент из выделенной области
int getMinInArea(matrix m) {
  position maxPos = getMaxValuePos(m);
  int localMin = m.values[maxPos.rowIndex][maxPos.colIndex];
  for (int i = maxPos.rowIndex - 1; i >= 0; i--) {
    int d = maxPos.rowIndex - i;
    for (int j = 0; j < 2 * d + 1; j++) {
      if (maxPos.colIndex - d + j < 0) continue;
      else if (maxPos.colIndex - d + j >= m.nCols) break;
      localMin = min(localMin, m.values[i][maxPos.colIndex - d + j]);
    }
  }
  return localMin;
}

// Задание 9

// Возвращает расстояние от точки находящейся в n-мерном пространстве, до
центра
float getDistance(int *a, int n) {
  float s = 0;
  for (int i = 0; i < n; i++)
    s += pow(a[i], 2);
  return sqrt(s);
}
```

```c
// Сортиурует матрицу m, по заданному критерию (criteria)
void insertionSortRowsMatrixByRowCriteriaF(
  matrix *m, float (*criteria)(int *, int)) {
  float rowsCriteria[m->nRows];
  for (int i = 0; i < m->nRows; i++)
    rowsCriteria[i] = criteria(m->values[i], m->nCols);

  for (int i = 1; i < m->nRows; i++)
    for (int j = i; j >= 1; j--)
      if (rowsCriteria[j] < rowsCriteria[j - 1]) {
        swapF(&rowsCriteria[j], &rowsCriteria[j - 1]);
        swapRows(m, j, j - 1);
      } else break;
}

// Сортирует матрицу m, по отдалению координат точки до центра
void sortByDistances(matrix *m) {
  insertionSortRowsMatrixByRowCriteriaF(m, getDistance);
}

// Задание 10

// Определить количество классов эквивалентных строк данной прямоугольной
// матрицы.
int countEqClassesByRowsSum(matrix m) {
  long long a[m.nRows];
  for (int i = 0; i < m.nRows; i++)
    a[i] = sumArray(m.values[i], m.nCols);
  return countNUnique(a, m.nRows);
}

// Задание 11

// Возвращает k – количество "особых" элементов матрицы, считая
// элемент "особым", если он больше суммы остальных элементов своего столбца.
int getNSpecialElement(matrix m) {
  int count = 0;
  for (int i = 0; i < m.nCols; i++) {
    int max = m.values[0][i];
    int sum = 0;
    for (int j = 1; j < m.nRows; j++) {
      if (m.values[j][i] > max) {
        sum += max;
        max = m.values[j][i];
      } else {
        sum += m.values[j][i];
      }
    }
    if (max > sum)
      count++;
  }
  return count;
}

// Задание 12

// Возвращает позицию левого минимального элемента
position getLeftMin(matrix m) {
  position min = {0, 0};
  for (int i = 0; i < m.nRows; i++)
    for (int j = 0; j < m.nCols; j++)
      if (m.values[i][j] < m.values[min.rowIndex][min.colIndex])
        min = (position) {i, j};
  return min;
}
```

```
// Заменяет предпоследнюю строку матрицы первым из столбцов, в котором
// находится минимальный элемент матрицы.
void swapPenultimateRow(matrix *m, int n) {
  position minPos = getLeftMin(*m);
  int a[n];
  for (int i = 0; i < n; i++)
    a[i] = m->values[i][minPos.colIndex];
  for (int i = 0; i < n; i++)
    m->values[n - 2][i] = a[i];
}

bool isNonDescendingSorted(int const *const a, int const n) {
  for (int i = 1; i < n; i++)
    if (a[i - 1] > a[i])
      return false;
  return true;
}

// Возвращает (true), если строки матрицы отсортированы по неубыванию,
// (false) - в ином случае
bool hasAllNonDescendingRows(matrix m) {
  for (int i = 0; i < m.nRows; i++)
    if (isNonDescendingSorted(m.values[i], m.nCols) == false)
      return false;
  return true;
}

// Возвращает (true), если все строки матрицы отсортированы по неубыванию,
// (false) - в ином случае
int countNonDescendingRowsMatrices(matrix *ms, int nMatrix) {
  for (int i = 0; i < nMatrix; i++)
    if (hasAllNonDescendingRows(ms[i]) == false)
      return false;
  return true;
}

// Задание 14

// Возвращает количество значений value в массиве a размера n
int countValues(const int *a, int n, int value) {
  int count = 0;
  for (int i = 0; i, n; i++)
    if (a[i] == value)
      count++;
  return count;
}

// Возвращает количество нулевых строк
int countZeroRows(matrix m) {
  int count = 0;
  for (int i = 0; i < m.nRows; i++)
    if (countValues(m.values[i], m.nCols, 0) == m.nCols)
      count++;
  return count;
}
```

```c
// Выводит матрицу с наибольшим количеством нулевых строк, среди массива
// матриц ms размера Matrix
void printMatrixWithMaxZeroRows(matrix *ms, int nMatrix) {
  matrix *msRes = getMemArrayOfMatrices(nMatrix, ms[0].nRows, ms[0].nCols);
  int count = 0;
  int max = 0;
  for (int i = 0; i < nMatrix; i++) {
    int countZero = countZeroRows(ms[i]);
    if (countZero == max) {
      msRes[count++] = ms[i];
    } else if (countZero > max) {
      count = 1;
      max = countZero;
      msRes[0] = ms[i];
    }
  }
  if (max != 0)
    outputMatrices(ms, count);
  freeMemMatrices(msRes, nMatrix);
}

// Возвращает максимум абсолютных величин матрицы m
int getNorm(matrix m) {
  int max = m.values[0][0];
  for (int i = 0; i < m.nRows; i++)
    for (int j = 0; j < m.nCols; j++)
      if (m.values[i][j] > max)
        max = m.values[i][j];
  return max;
}

// Выводит матрицу с наименьшей нормой
void minimOfNorms(matrix *ms, int nMatrix) {
  matrix *msRes = getMemArrayOfMatrices(nMatrix, ms[0].nRows, ms[0].nCols);
  int minNorm = getNorm(ms[0]);
  int count = 0;
  for (int i = 1; i < nMatrix; i++) {
    int norm = getNorm(ms[i]);
    if (minNorm > norm) {
      minNorm = norm;
      count = 0;
    } else if (minNorm == norm) {
      msRes[count++] = ms[i];
    }
  }
  outputMatrices(msRes, count);
  freeMemMatrices(msRes, nMatrix);
}
```

Тесты:

```c
#include <stdio.h>
#include "libs/data_structures/matrix/matrix.h"
#include "libs/algorithms/algorithms.h"

void test_transposeSquareMatrix() {
  matrix m = createMatrixFromArray((int[]) {1, 2, 3,
                                            4, 5, 6,
                                            7, 8, 9}, 3, 3);
  transposeSquareMatrix(&m);
  matrix res = createMatrixFromArray((int[]) {1, 4, 7,
                                              2, 5, 8,
                                              3, 6, 9}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_exchangeMaxAndMinRow() {
  matrix m = createMatrixFromArray((int[]) {1, 2, 3,
                                            4, 5, 6,
                                            7, 8, 9}, 3, 3);
  exchangeMaxAndMinRow(&m);
  matrix res = createMatrixFromArray((int[]) {7, 8, 9,
                                              4, 5, 6,
                                              1, 2, 3}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_sortRowsByMinElement() {
  matrix m = createMatrixFromArray((int[]) {7, 1, 2,
                                            1, 8, 1,
                                            3, 2, 3}, 3, 3);
  sortRowsByMaxElement(&m);
  matrix res = createMatrixFromArray((int[]) {7, 1, 2,
                                              3, 2, 3,
                                              1, 8, 1}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_sortColsByMinElement() {
  matrix m = createMatrixFromArray((int[]) {7, 5, 3,
                                            9, 2, 3,
                                            3, 6, 3}, 3, 3);
  sortColsByMinElement(&m);
  matrix res = createMatrixFromArray((int[]) {7, 3, 5,
                                              9, 3, 2,
                                              3, 3, 6}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_mulMatrices() {
  matrix m1 = createMatrixFromArray((int[]) {1, 2, 3, 4,
                                             5, 6, 7, 8,
                                             9, 10, 11, 12}, 3, 4);
```

```
  matrix m2 = createMatrixFromArray((int[]) {1, 2, 3, 4,
                                             5, 6, 7, 8,
                                             9, 10, 11, 12}, 4, 3);
  matrix res = mulMatrices(m1, m2);
  matrix test = createMatrixFromArray(
    (int[]) {70, 80, 90, 158, 184, 210, 246, 288, 330}, 3, 3);

  assert(twoMatricesEqual(res, test));
  freeMemMatrix(m1);
  freeMemMatrix(m2);
  freeMemMatrix(res);
  freeMemMatrix(test);
}

void test_getSquareOfMatrixIfSymmetric() {
  matrix m = createMatrixFromArray((int[]) {1, 1, 2,
                                            1, 2, 3,
                                            2, 3, 3}, 3, 3);
  getSquareOfMatrixIfSymmetric(&m);
  matrix res = createMatrixFromArray((int[]) {6, 9, 11,
                                              9, 14, 17,
                                              11, 17, 22}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_transposeIfMatrixHasNotEqualSumOfRows() {
  matrix m = createMatrixFromArray((int[]) {1, 2, 3,
                                            4, 5, 6,
                                            7, 8, 9}, 3, 3);
  transposeSquareMatrix(&m);
  matrix res = createMatrixFromArray((int[]) {1, 4, 7,
                                              2, 5, 8,
                                              3, 6, 9}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_isMutuallyInverseMatrices() {
  matrix m1 = createMatrixFromArray((int[]) {3, -5,
                                             1, -2}, 2, 2);
  matrix m2 = createMatrixFromArray((int[]) {2, -5,
                                             1, -3}, 2, 2);

  assert(isMutuallyInverseMatrices(m1, m2));
  freeMemMatrix(m1);
  freeMemMatrix(m2);
}

void test_getMinInArea_zoneDoesNotGoBeyond() {
  matrix m = createMatrixFromArray((int[]) {3, 2, 3,
                                            4, 10, 6,
                                            7, 8, 9}, 3, 3);

  assert(getMinInArea(m) == 2);
  freeMemMatrix(m);
}
```

```
void test_getMinInArea_zoneOverflowsOnTheLeft() {
  matrix m = createMatrixFromArray((int[]) {3, 2, 3,
                                            10, 1, 6,
                                            7, 8, 9}, 3, 3);

  assert(getMinInArea(m) == 2);
  freeMemMatrix(m);
}

void test_getMinInArea_zoneOverflowsOnTheRight() {
  matrix m = createMatrixFromArray((int[]) {9, 8, 9,
                                            1, 1, 10,
                                            7, 8, 9}, 3, 3);

  assert(getMinInArea(m) == 8);
  freeMemMatrix(m);
}

void test_getMinInArea_zoneIsOutOfBounds() {
  matrix m = createMatrixFromArray((int[]) {9, 8, 9,
                                            10, 1, -10,
                                            7, 81, 9}, 3, 3);

  assert(getMinInArea(m) == -10);
  freeMemMatrix(m);
}

void test_getMinInArea() {
  test_getMinInArea_zoneDoesNotGoBeyond();
  test_getMinInArea_zoneOverflowsOnTheLeft();
  test_getMinInArea_zoneOverflowsOnTheRight();
  test_getMinInArea_zoneIsOutOfBounds();
}

void test_sortByDistances() {
  matrix m = createMatrixFromArray((int[]) {3, -5, 2,
                                            4, 6, -3,
                                            1, 3, 0}, 3, 3);
  sortByDistances(&m);
  matrix res = createMatrixFromArray((int[]) {1, 3, 0,
                                              3, -5, 2,
                                              4, 6, -3}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_getNSpecialElement() {
  matrix m = createMatrixFromArray((int[]) {3, 5, 5, 4,
                                            2, 3, 6, 7,
                                            12, 2, 1, 2}, 3, 4);

  assert(getNSpecialElement(m) == 2);
  freeMemMatrix(m);

}
```

```
void test_swapPenultimateRow() {
  matrix m = createMatrixFromArray((int[]) {1, 2, 3,
                                            4, 5, 6,
                                            7, 8, 1}, 3, 3);
  swapPenultimateRow(&m, m.nCols);
  matrix res = createMatrixFromArray((int[]) {1, 2, 3,
                                              1, 4, 7,
                                              7, 8, 1}, 3, 3);

  assert(twoMatricesEqual(m, res));
  freeMemMatrix(m);
  freeMemMatrix(res);
}

void test_countNonDescendingRowsMatrices() {

}

void test() {
  test_transposeSquareMatrix();
  test_exchangeMaxAndMinRow();
  test_sortRowsByMinElement();
  test_sortColsByMinElement();
  test_mulMatrices();
  test_getSquareOfMatrixIfSymmetric();
  test_transposeIfMatrixHasNotEqualSumOfRows();
  test_isMutuallyInverseMatrices();
  test_getMinInArea();
  test_sortByDistances();
  test_getNSpecialElement();
  test_swapPenultimateRow();
  test_countNonDescendingRowsMatrices();
}

int main() {
  test();

  return 0;
}
```

```
commit 89e6da0997db3663ce057d53b9e3a6486989b56c (HEAD -> master, origin/master)
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Wed Feb 16 01:15:34 2022 +0300

    1/2 final

 libs/data_structures/matrix/matrix.c | 14 ++++++++++++++
 1 file changed, 14 insertions(+)

commit b2d69f34725015eec30ca6ddc17fb774ee6d2c0b
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 23:57:12 2022 +0300

    realization: findSumOfMaxesOfPseudoDiagonal

 libs/data_structures/matrix/matrix.c | 20 ++++++++++++++++++++-
 1 file changed, 19 insertions(+), 1 deletion(-)

commit adfbdd85ad6669575b79db055512d08c205fa1b5
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 23:06:26 2022 +0300

    realization: isMutuallyInverseMatrices

 libs/data_structures/matrix/matrix.c | 4 ++++
 1 file changed, 4 insertions(+)

commit 332fc7a9858c0c60f82bd1fd143b970c53fa270e
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 23:01:32 2022 +0300

    realization: transposeIfMatrixHasNotEqualSumOfRows

 libs/data_structures/matrix/matrix.c | 13 ++++++++++++--
 1 file changed, 11 insertions(+), 2 deletions(-)

commit 2252052dfd4ea4b3be3324b170f457ae7181febe
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 22:41:55 2022 +0300

    refactoring: getSquareOfMatrixIfSymmetric

 libs/data_structures/matrix/matrix.c | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

commit da7630e42a643b77a83395f5c9a100128a604a0c
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 21:27:36 2022 +0300

    realization: getSquareOfMatrixIfSymmetric

 libs/data_structures/matrix/matrix.c | 24 ++++++++++++++++++++++---
 1 file changed, 21 insertions(+), 3 deletions(-)
```

```
commit 09a0e5e79e1cbd632d86f952fb6ec776b8852f39
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 20:24:48 2022 +0300

    realization: sortColsByMinElement

 libs/data_structures/matrix/matrix.c | 37 ++++++++++++++++++++----------------
 1 file changed, 22 insertions(+), 15 deletions(-)

commit 6fc216bb710240b1709a188fd5d58b5d9753a187
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 19:31:02 2022 +0300

    realization: sortRowsByMaxElement

 libs/data_structures/matrix/matrix.c | 27 +++++++++++++++++++++++++----
 1 file changed, 23 insertions(+), 4 deletions(-)

commit eb18ee856041626ae982a3c4d075d733b451a5a8
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 15:01:42 2022 +0300

    realization: exchangeMaxAndMinRow

 libs/data_structures/matrix/matrix.c | 38 ++++++++++++++++++++++++++++++++++++++---
 1 file changed, 35 insertions(+), 3 deletions(-)

commit 6ebde973173b046e48260df575275a2afa6cba5c
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 14:46:27 2022 +0300

    realization: max/min

 libs/data_structures/matrix/matrix.c | 9 +++++++++
 1 file changed, 9 insertions(+)

commit ce1bb65d00c4650763e1eed638f793a7786bf4b5
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 14:44:36 2022 +0300

    realization: transpose

 libs/data_structures/matrix/matrix.c | 18 +++++++++++++---------
 1 file changed, 9 insertions(+), 9 deletions(-)

commit ff978c2021f6e17b89a237e0882d244f389b4f76
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 14:32:26 2022 +0300

    realization: transpose

 libs/data_structures/matrix/matrix.c | 9 ++++++++-
 1 file changed, 8 insertions(+), 1 deletion(-)
```

```
commit cefd50f14c7234ecb5e8d85c5b5bdfad42587580
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 13:44:46 2022 +0300

    realization: isMatrix

 libs/data_structures/matrix/matrix.c | 42 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
 1 file changed, 41 insertions(+), 1 deletion(-)

commit 4fe40635f06213925984f9f479adc28de750ed79
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Tue Feb 15 10:18:19 2022 +0300

    realization: sort

 libs/data_structures/matrix/matrix.c | 11 +++++++++++
 1 file changed, 11 insertions(+)

commit 8d7353b675788fabca4906a6f98d9f8ef09f4f59
Author: ChuvilkoDEV <ilyasutton@gmail.com>
Date:   Mon Feb 14 13:34:57 2022 +0300

    realization: swap columns/rows

 libs/data_structures/matrix/matrix.c | 75 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 75 insertions(+)
```