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 №16**

по дисциплине: Основы программирования

тема: «Работа с многомерными массивами»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

Притчин Иван Сергеевич

Черников Сергей Викторович

Код-ревьер: ст. группы ПВ-223

Голуцкий Георгий Юрьевич

Белгород 2023 г.

**Лабораторная работа № 16**

**Вариант 1**

**Содержание отчёта:**

* Тема лабораторной работы.
* Цель лабораторной работы.
* Решения задач.
  + Текст задания.
  + Исходный код (в т.ч. и тестов).
* Ссылка на открытый репозиторий с решением.
* Скриншот с историей коммитов.
* Работа над ошибками (код ревью)
* Вывод по работе.

**Тема лабораторной работы:** Работа с многомерными массивами

**Цель лабораторной работы:** получение навыков работы с многомерными массивами.

**Решения задач:**

1. Дана квадратная матрица. Если среди сумм элементов строк матрицы нет равных, то транспонировать матрицу.
   * bool isUnique(long long \*a, int n)
   * long long getSum(int \*a, int n)
   * void transposeIfMatrixHasNotEqualSumOfRows(matrix m)

libs/alg/lab16/5func.c

#include "../alg.h"  
  
**bool** isUnique(long long \*a, int n) {  
 for (int i = 0; i < n - 1; i++)  
 for (int j = i + 1; j < n; j++)  
 if (a[i] == a[j])  
 return **false**;  
  
 return **true**;  
}  
  
long long getSum(int \*a, int n) {  
 long long sum = 0;  
  
 for (int i = 0; i < n; i++)  
 sum += a[i];  
  
 return sum;  
}

libs/data\_structures/matrix.c

void transposeIfMatrixHasNotEqualSumOfRows(Matrix m) {  
 long long \*rowSums = (long long \*) malloc(sizeof(long long) \* m.nRows);  
  
 for (int i = 0; i < m.nRows; i++)  
 rowSums[i] = getSum(m.values[i], m.nCols);  
  
 if (isUnique(rowSums, m.nRows))  
 transposeSquareMatrix(m);  
  
 free(rowSums);  
}

lab16/main.c (testTransposeIfMatrixHasNotEqualSumOfRows)

void testTransposeIfMatrixHasNotEqualSumOfRows() {  
 Matrix m = createMatrixFromArray((int[]) {1, 1, 1,  
 2, 2, 2,  
 3, 3, 3}, 3, 3);  
 transposeIfMatrixHasNotEqualSumOfRows(m);  
  
 Matrix expectedMatrix = createMatrixFromArray((int[]) {1, 2, 3,  
 1, 2, 3,  
 1, 2, 3}, 3, 3);  
  
 **assert**(areTwoMatricesEqual(m, expectedMatrix));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedMatrix);  
  
 m = createMatrixFromArray((int[]) {1, 2, 1,  
 2, 1, 1,  
 3, 3, 3}, 3, 3);  
 transposeIfMatrixHasNotEqualSumOfRows(m);  
  
 expectedMatrix = createMatrixFromArray((int[]) {1, 2, 1,  
 2, 1, 1,  
 3, 3, 3}, 3, 3);  
  
 **assert**(areTwoMatricesEqual(m, expectedMatrix));  
  
 freeMemMatrix(m);  
 freeMemMatrix(expectedMatrix);  
}

1. Даны две квадратные матрицы 𝐴 и 𝐵. Определить, являются ли они взаимно обратными (𝐴 = 𝐵−1).
   * Matrix mulMatrices(Matrix m1, Matrix m2)
   * bool isMutuallyInverseMatrices(Matrix m1, Matrix m2)

libs/data\_structures/matrix.c

Matrix mulMatrices(Matrix m1, Matrix m2) {  
 **assert**(isSquareMatrix(m1) && isSquareMatrix(m2) && m1.nRows == m2.nRows);  
  
 Matrix multipliedMatrix = getMemMatrix(m1.nRows, m1.nRows);  
  
 for (int i = 0; i < m1.nRows; i++) {  
 for (int j = 0; j < m1.nCols; j++) {  
 int arrayElement = 0;  
  
 for (int k = 0; k < m1.nCols; k++)  
 arrayElement += m1.values[i][k] \* m2.values[k][j];  
  
 multipliedMatrix.values[i][j] = arrayElement;  
 }  
 }  
  
 return multipliedMatrix;  
}  
  
**bool** isMutuallyInverseMatrices(Matrix m1, Matrix m2) {  
 Matrix multiplication = mulMatrices(m1, m2);  
  
 **bool** isMutuallyInversed = isEMatrix(multiplication);  
  
 freeMemMatrix(multiplication);  
  
 return isMutuallyInversed;  
}

lab16/main.c (testIsMutuallyInverseMatrices)

void testIsMutuallyInverseMatrices() {  
 Matrix m1 = createMatrixFromArray((int[]) {2, 5, 7,  
 6, 3, 4,  
 5, -2, -3}, 3, 3);  
  
 Matrix m2 = createMatrixFromArray((int[]) {1, -1, 1,  
 -38, 41, -34,  
 27, -29, 24}, 3, 3);  
  
 **assert**(isMutuallyInverseMatrices(m1, m2));  
  
 freeMemMatrix(m1);  
 freeMemMatrix(m2);  
  
 m1 = createMatrixFromArray((int[]) {42, 5, 71,  
 62, 6, 4,  
 5, -2, 33}, 3, 3);  
  
 m2 = createMatrixFromArray((int[]) {1, 1, 1,  
 -3, 41, -6,  
 27, -292, 24}, 3, 3);  
  
 **assert**(!isMutuallyInverseMatrices(m1, m2));  
  
 freeMemMatrix(m1);  
 freeMemMatrix(m2);  
}

1. Дан массив матриц одного размера. Определить число матриц, строки которых упорядочены по неубыванию элементов
   * bool isNonDescendingSorted(int \*a, int n)
   * bool hasAllNonDescendingRows(Matrix m)
   * int countNonDescendingRowsMatrices(Matrix \*ms, int nMatrix)

libs/alg/lab16/13func.c

#include "../alg.h"  
  
**bool** isNonDescendingSorted(int \*a, int n) {  
 for (int i = 0; i < n - 1; i++)  
 if (a[i] > a[i + 1])  
 return **false**;  
  
 return **true**;  
}

libs/data\_structures/matrix.c

**bool** hasAllNonDescendingRows(Matrix m) {  
 for (int i = 0; i < m.nRows; i++) {  
 if (!isNonDescendingSorted(m.values[i], m.nCols))  
 return **false**;  
 }  
  
 return **true**;  
}  
  
int countNonDescendingRowsMatrices(Matrix \*ms, int nMatrix) {  
 int counter = 0;  
  
 for (int i = 0; i < nMatrix; i++)  
 counter += hasAllNonDescendingRows(ms[i]);  
  
 return counter;  
}

lab16/main.c (testCountNonDescendingRowsMatrices)

void testCountNonDescendingRowsMatrices() {  
 Matrix \*matrices = createArrayOfMatrixFromArray((int[]){7, 1,  
 1, 1,  
  
 1, 6,  
 2, 2,  
  
 5,4,  
 2,3,  
  
 1,3,  
 7,9}, 4, 2, 2);  
  
 **assert**(countNonDescendingRowsMatrices(matrices, 4) == 2);  
  
 freeMemMatrices(matrices, 4);  
  
 matrices = createArrayOfMatrixFromArray((int[]){3, 3,  
 6, 2,  
 42, 2,  
  
 1, 6,  
 2, 2,  
 8, 9,  
  
 1, 1,  
 2, 2,  
 3, 3,  
  
 1,3,  
 9,7,  
 7, 7}, 4, 3, 2);  
  
 **assert**(countNonDescendingRowsMatrices(matrices, 4) == 2);  
  
 freeMemMatrices(matrices, 4);  
}

1. Дан массив целочисленных квадратных матриц. Вывести матрицы с наименьшей нормой. В качестве нормы матрицы взять максимум абсолютных величин ее элементов.

libs/alg/lab16/15func.c

#include "../alg.h"  
  
int intMin2(int a, int b) {  
 return a < b ? a : b;  
}

libs/data\_structures/matrix.c

int countNorm(Matrix m) {  
 *// We can do this since we compare values of matrix by modulus* int max = 0;  
  
 for (int i = 0; i < m.nRows; i++)  
 for (int j = 0; j < m.nCols; j++)  
 max = max2(max, abs(m.values[i][j]));  
  
 return max;  
}  
  
void outputMatricesWithMinNorm(Matrix \*ms, int nMatrix) {  
 if (nMatrix == 0) return;  
  
 int \*keys = (int \*) malloc(sizeof(int) \* nMatrix);  
 int minKey = countNorm(ms[0]);  
 keys[0] = minKey;  
  
 for (int i = 1; i < nMatrix; i++) {  
 **assert**(isSquareMatrix(ms[i]));  
  
 keys[i] = countNorm(ms[i]);  
 minKey = intMin2(minKey, keys[i]);  
 }  
  
 for (int i = 0; i < nMatrix; i++)  
 if (keys[i] == minKey)  
 outputMatrix(ms[i]);  
  
 free(keys);  
}

1. Каждая строка данной матрицы представляет собой координаты вектора в пространстве. Определить, какой из этих векторов образует максимальный угол с данным вектором 𝑣.
   * double getScalarProduct(int \*a, int \*b, int n)
   * double getVectorLength(int \*a, int n)
   * double getCosine(int \*a, int \*b, int n)
   * int getVectorIndexWithMaxAngle(Matrix m, int \*b)

libs/alg/lab16/17func.c

#include "../alg.h"  
  
double getScalarProduct(int \*a, int \*b, int n) {  
 int scalarProduct = 0;  
  
 for (int i = 0; i < n; i++)  
 scalarProduct += a[i] \* b[i];  
  
 return scalarProduct;  
}  
  
double getVectorLength(int \*a, int n) {  
 int sumOfCubes = 0;  
  
 for (int i = 0; i < n; i++)  
 sumOfCubes += a[i] \* a[i];  
  
 return sqrt(sumOfCubes);  
}  
  
double getCosine(int \*a, int \*b, int n) {  
 double vecLenA = getVectorLength(a, n);  
 double vecLenB = getVectorLength(a, n);  
  
 *// The angle between zero-vector(s) is 0 degree, cos(0) = 1* if (vecLenA == 0 || vecLenB == 0) return 1;  
  
 return getScalarProduct(a, b, n) / (vecLenA \* vecLenB);  
}

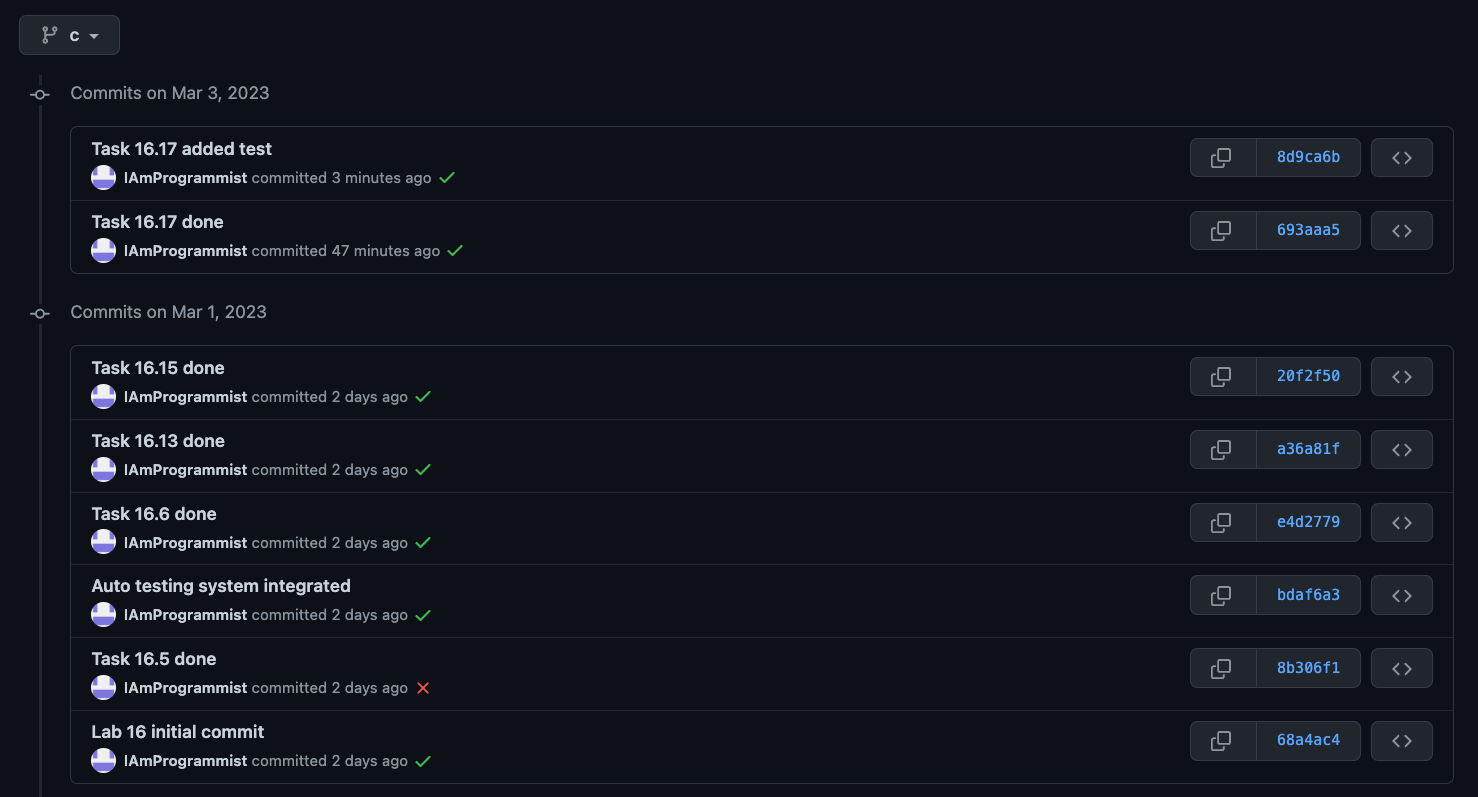
libs/data\_structures/matrix.c

int getVectorIndexWithMaxAngle(Matrix m, int \*b) {  
 if (m.nRows == 0) return -1;  
  
 double minCosinus = getCosine(m.values[0], b, m.nCols);  
 int minCosinusIndex = 0;  
  
 for (int i = 1; i < m.nRows; i++) {  
 double currentCos = getCosine(m.values[i], b, m.nCols);  
  
 *// More cos - less angle* if (currentCos < minCosinus) {  
 minCosinus = currentCos;  
 minCosinusIndex = i;  
 }  
 }  
  
 return minCosinusIndex;  
}

lab16/main.c (testGetVectorIndexWithMaxAngle)

void testGetVectorIndexWithMaxAngle() {  
 Matrix vectors = createMatrixFromArray((int[]){1, 3,  
 -2, 3,  
 -2, -3,  
 -4,-2, 4, -2,  
 }, 5, 2);  
  
 int vector[2] = {6, 1};  
  
 **assert**(getVectorIndexWithMaxAngle(vectors, vector) == 3);  
  
 freeMemMatrix(vectors);  
  
 vectors = createMatrixFromArray((int[]){0, 0,  
 0, 0,  
 0, 0}, 3, 2);  
  
 **assert**(getVectorIndexWithMaxAngle(vectors, vector) == 0);  
  
 freeMemMatrix(vectors);  
  
 vectors = createMatrixFromArray((int[]){}, 0, 2);  
  
 **assert**(getVectorIndexWithMaxAngle(vectors, vector) == -1);  
  
 freeMemMatrix(vectors);  
}

Ссылка на репозиторий: <https://github.com/IAmProgrammist/programming-and-algorithmization-basics>



**Вывод:** в ходе выполнения лабораторной работы получены навыки работы с многомерными массивами.