# Задание

1. Операции над матрицами

а) Сортировка одномерного массива (реализовать 2 алгоритма сортировки и исследовать возможности их оптимизации).

б) Вычисление детерминанта, поиск максимума по строкам, минимума по столбцам.

в) Нахождение обратной матрицы, выполнение умножения матриц А и В.

Сортировка одномерного массива

В программе реализованы два алгоритма сортировки массива: и быстрая сортировка Хоара и пирамидальная сортировка.

# Быстрая сортировка

Общая идея алгоритма состоит в следующем:

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива или же число, вычисленное на основе значений элементов. На практике обычно выбирается средний элемент массива.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие».
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Быстрая сортировка Хоара

\*data - массив элементов

left - номер первого элемента сортируемого массива

right - номер последнего элемента сортируемого массива

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void quickSort(double \*data, int left, int right)

{

int left1 = left;

int right1 = right;

double p;

double ref = data[(left+right)/2];

do{

while(data[left1] < ref)left1++;

while(data[right1] > ref)right1--;

if(left1<=right1){

if(data[left1] > data[right1])

data[left1] = (p = data[right1], data[right1] = data[left1], p);

left1++;

right1--;

}

}while (left1 <= right1);

if(left1 < right)

quickSort(data, left1, right);

if(left < right1)

quickSort(data, left, right1);

}

# Контрольные примеры работы программы (сортировка Хоара)

Пример 1.

Рассмотрим сортировку на обычном массиве (рис. 1).

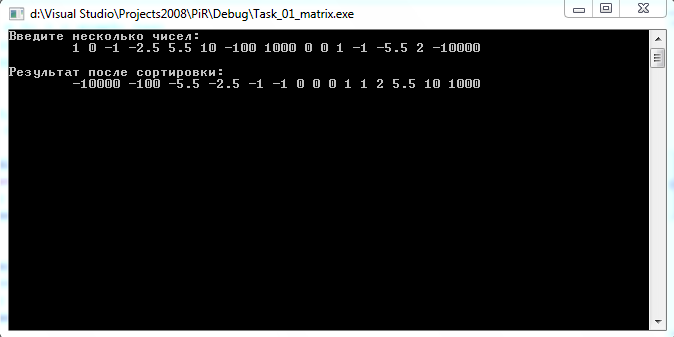
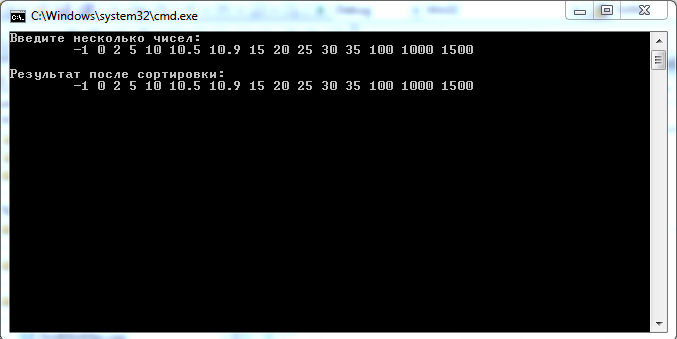


Рис.1. Результат быстрой сортировки на массиве из 15 элементов.

Пример 2.

Все элементы находятся на своих местах (рис. 2).

Рис.2. Результат быстрой сортировки на упорядоченном массиве из 15 элементов.

Пример 3.

Сортировка массива из элементов, расположенных по убыванию (в обратном порядке) (рис. 3).

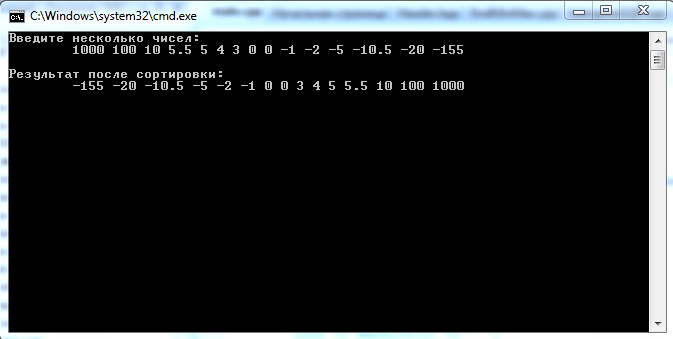


Рис.3. Результат быстрой сортировки на массиве из 15 элементов, расположенных по убыванию.

Пример 4.

Сортировка одного элемента (рис. 4)

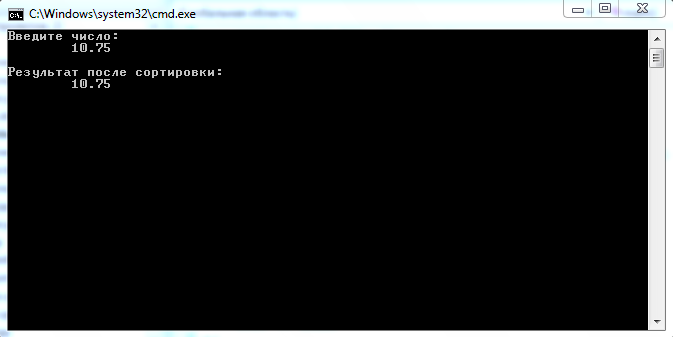


Рис.4. Результат быстрой сортировки на массиве из одного элемента.

# Пирамидальная сортировка

Сортировка пирамидой использует бинарное сортирующее дерево. Сортирующее дерево — это такое дерево, у которого выполнены условия:

1. Каждый лист имеет глубину либо d, либо d-1, d — максимальная глубина дерева.
2. Значение в любой вершине не меньше значения её потомков.

Удобная структура данных для сортирующего дерева — такой массив Array, что Array[0] — элемент в корне, а потомки элемента Array[i] являются Array[2i+1] и Array[2i+2]. Далее будем рассматривать только эту структуру.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Пирамидальная сортировка

data - массив элементов

length - длина массива

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void heapSort(double \*data, int length)

{

double swap;

for(int j = length-1; j > 0; --j)

{

for(int i = j/2; i >= 0; i--)

{

if(data[i] < data[2\*i+1] && 2\*i+1 <= j)

data[i] = (swap = data[2\*i+1], data[2\*i+1] = data[i], swap);

if(data[i] < data[2\*i+2] && 2\*i+2 <= j)

data[i] = (swap = data[2\*i+2], data[2\*i+2] = data[i], swap);

}

data[0] = (swap = data[j], data[j] = data[0], swap);

}

}

# Контрольные примеры работы программы (сортировка Хоара)

Пример 1.

Рассмотрим сортировку на обычном массиве (рис. 5).

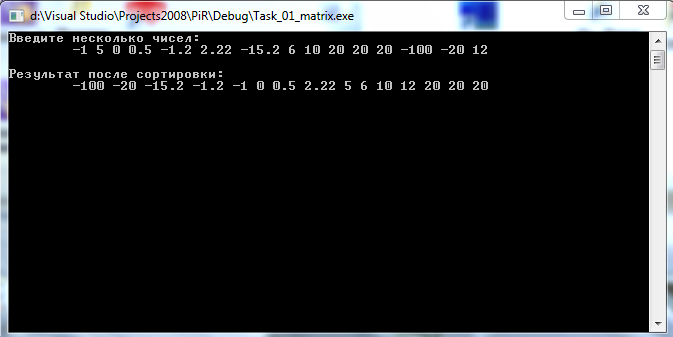


Рис.5. Результат пирамидальной сортировки на массиве из 15 элементов.

Пример 2.

Сортировка упорядоченного по возрастанию массива (рис. 6).

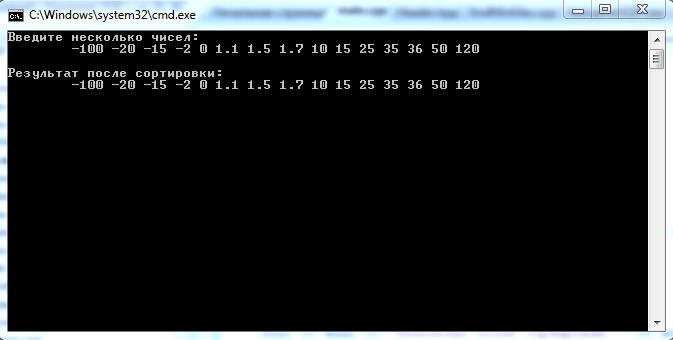


Рис.6. Результат пирамидальной сортировки на упорядоченном массиве из 15 элементов.

Пример 3.

Сортировка упорядоченного по убыванию массива (рис. 7).

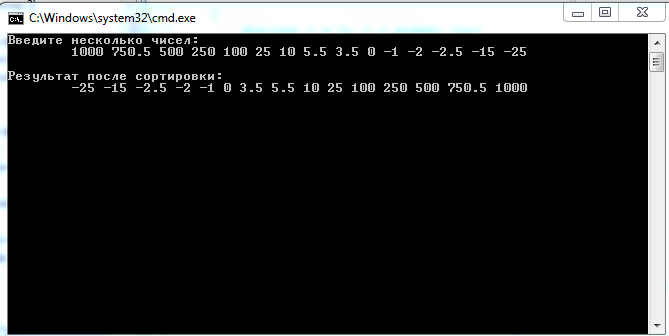


Рис.7. Результат пирамидальной сортировки на упорядоченном по убыванию массиве из 15 элементов.

Пример 4.

Сортировка одного элемента (рис. 8).

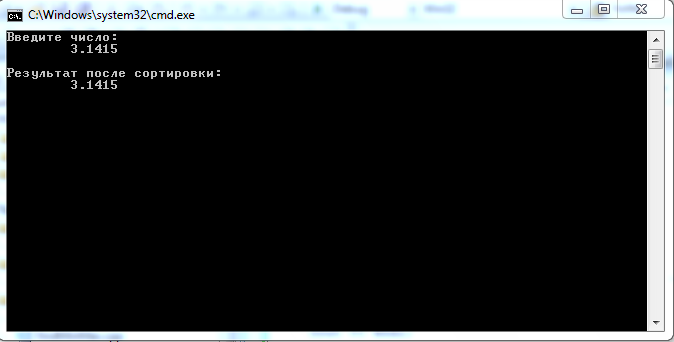


Рис.8. Результат пирамидальной сортировки на массиве из одного элемента.

# Матрицы. Операции над матрицами.

Будем представлять матрицу в виде двумерного массива чисел двойной точности.

# Вычисление определителя.

Определитель будем вычислять, используя формулу разложения по строке:

https://upload.wikimedia.org/math/1/2/f/12f60bbd0049cc8c8b58f162c80c9a44.png , где \bar M_j^1 — [дополнительный минор](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B8%D0%BD%D0%BE%D1%80) к элементу a_{1j}.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Определитель квадратной матрицы

size - сторона матрицы

\*\*data - матрица

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double det(int size, double \*\*data){

int str, col, currCol;

double result = 0;//результат выполнения (детерминант)

if(size == 1)

return data[0][0];

double \*\*newData = new double\*[size-1];

for(str = 0; str < size-1; str++)

newData[str] = new double[size-1];

//Создание новой матрицы из старой, выкидыванием столбца col и строки str

for(col = 0; col < size; col++){

for(currCol = 0; currCol < size; currCol++){

if(currCol != col)

for(str = 1; str < size; str++){

newData[str-1][currCol-(currCol>col)] = data[str][currCol];

}

}

if(col%2)

result -= data[0][col]\*det(size-1, newData);

else

result += data[0][col]\*det(size-1, newData);

}

for(int i = 0; i < size-1; i++)

delete []newData[i];

delete []newData;

return result;

}

# Контрольные примеры работы программы (вычисление определителя квадратной матрицы)

Пример 1.

Вычисление определителя треугольной матрицы 5х5 (рис. 9). Как известно, он равен произведению элементов главной диагонали.

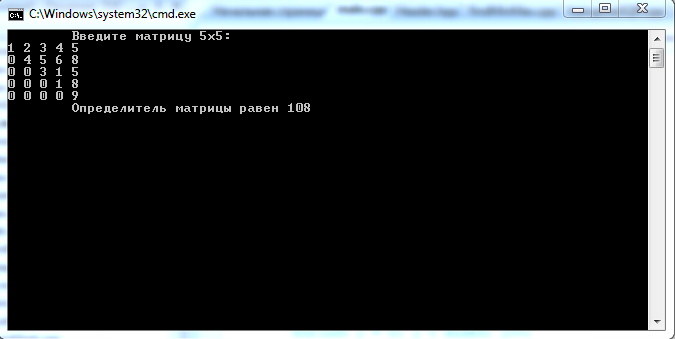
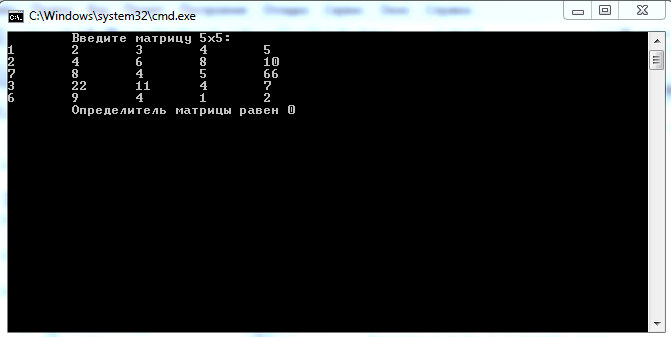


Рис.9. Вычисление определителя треугольной матрицы.

Пример 2.

Вычисление определителя с двумя линейно зависимыми строками (рис. 10). Как известно, он равен нулю.

 Рис.10. Вычисление определителя матрицы с линейно зависимыми строками (1-й и 2-й).

Пример 3.

Вычисление определителя обычной матрицы 2х2 (рис. 11).

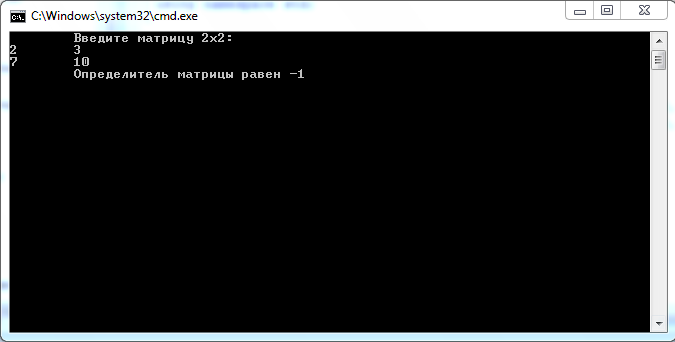
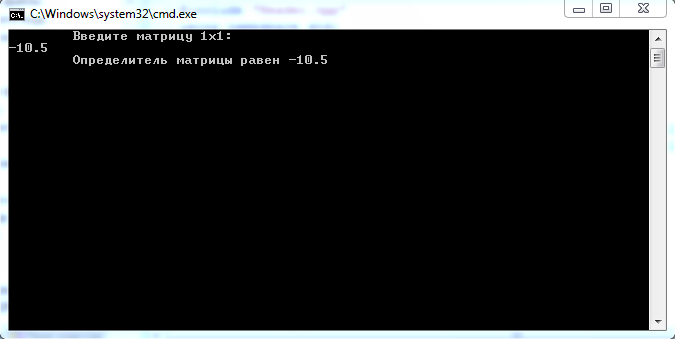


Рис.11. Вычисление определителя матрицы 2х2.

Пример 4.

Вычисление определителя матрицы 1х1 (рис. 12).

 Рис.12. Вычисление определителя матрицы 1х1.

# Поиск максимума по строкам, минимума по столбцам.

Искать максимальный элемент матрицы по строкам и минимальный по столбцам будем с помощью перебора всех элементов.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Поиск максимального элемента по строкам, минимального по столбцам

\*\*data - матрица

height - высота матрицы (кол-во строк)

weight - ширина матрицы (кол-во столбцов)

&min - минимальный элемент

&max - максимальный элемент

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void findMinMax

(double \*\*data, int height, int weight, double &min, double &max)

{

min = max = data[0][0];

//Поиск максимального элемента по строкам

for(int str = 0; str < height; str++)

for(int col = 0; col < weight; col++)

if(max < data[str][col])

max = data[str][col];

//Поиск минимального элемента по столбцам

for(int col = 0; col < weight; col++)

for(int str = 0; str < height; str++)

if(min > data[str][col])

min = data[str][col];

}

Пример (рис 13).

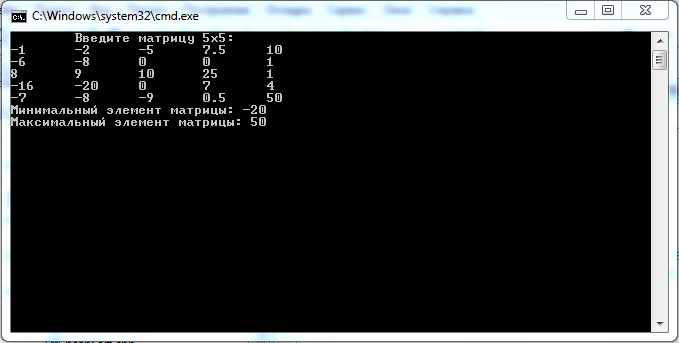


Рис.13. Нахождение минимального и максимального элементов.

# Умножение матриц.

Произведение матриц A \* B = C будем находить по формуле:

 c_{ij} = \sum_{r=1}^n a_{ir}b_{rj} \;\;\; \left(i=1, 2, \ldots m;\; j=1, 2, \ldots q \right).

При этом, матрицы А и B имеют размеры m x n и n x q соответственно.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Произведение матриц

\*\*data1 - первый множитель

\*\*data2 - второй множитель

size1 - кол-во строк в первой матрице

size2 - колво столбцов в первой матрице и кол-во строк во второй

size3 - кол-во столбцов во второй матрице

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double \*\*mult(double \*\*data1, double \*\*data2, const int size1, const int size2, const int size3){

double \*\* result;

result = new double \*[size1];//результат произведения

for(int i = 0; i < size3; i++)

result[i] = new double[size3];

for(int str = 0; str < size1; str++)//строки первой матрицы

{

for(int col = 0; col < size3; col++)//столбцы второй матрицы

{

result[str][col] = 0;

for(int i = 0; i < size2; i++)

result[str][col] += data1[str][i]\*data2[i][col];

}

}

return result;

}

# Контрольные примеры работы программы (умножение матриц)

Пример 1.

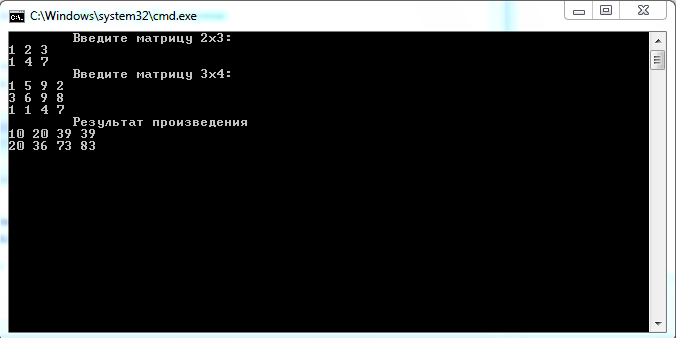


Рис.14. Произведение матриц 2х3 и 3х4.

Пример 2.

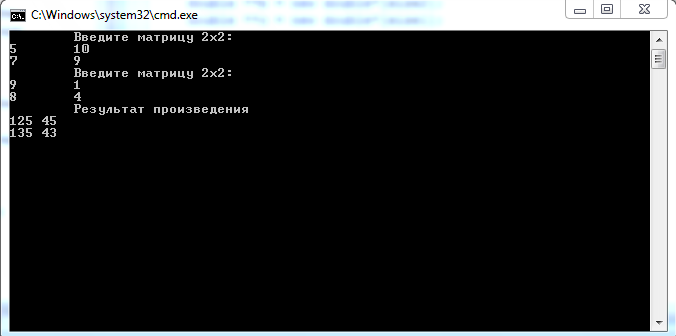


Рис.15. Произведение двух матриц 2х2.

# Нахождение обратной матрицы

Вычисление обратной матрицы будем выполнять с помощью алгебраических дополнений:

Пусть задана матрица An×n. Для того, чтобы найти элементы матрицы A−1, требуется осуществить три шага:

1. Найти определитель матрицы A и убедиться, что он не равен нулю 0, т.е. что матрица А – невырожденная.
2. Составить алгебраические дополнения Aij каждого элемента матрицы A и записать матрицу A∗n×n=(Aij) из найденных алгебраических дополнений.
3. Записать обратную матрицу с учетом формулы A−1= A∗)T, где Т – транспонирование.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление обратной матрицы (только для квадратных)

\*\*data - матрица

size - сторона матрицы

Функция возвращает NULL, если обратной матрицы не существует.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double \*\*inverse(double \*\*data, int size){

double \*\*additionMatrix;//Матрица алгебраических дополнений

double detData;

if((detData = det(size, data)) == 0)//вычисляем детерминант, если он равен нулю, то обратной матрицы не существует

return NULL;

additionMatrix = new double\*[size];

for(int str = 0; str < size; str++)

additionMatrix[str] = new double[size];

//Отдельный случай для матрицы 1х1

if(size == 1)

{

additionMatrix[0][0] = 1/data[0][0];

return additionMatrix;

}

//Составляем матрицу алгебраических дополнений

for(int str = 0; str < size; str++)

for(int col = 0; col < size; col++)

if(((str+col) % 2) == 0)

additionMatrix[str][col] = minor(data, size, str, col);

else

additionMatrix[str][col] = -minor(data, size, str, col);

transposeSQR(additionMatrix, size);//транспонируем матрицу

for(int str = 0; str < size; str++)//делим элементы на детерминант

for(int col = 0; col < size; col++)

additionMatrix[str][col] /= detData;

return additionMatrix;

}

Для реализации этой функции нам потребовались ещё две: вычисление минора матрицы для элемента Aij и транспонирование матрицы.

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление минора для квадратной матрицы

\*\*data - матрица

size - сторона матрицы

strin - строка, где располагается минор

colon - столбец, где располагается минор

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double minor(double \*\*data, int size, int strin, int colon){

double resultMinor;

double \*\*result = new double\*[size-1];

for(int i = 0; i < size-1; i++)

result[i] = new double[size-1];

//Создание новой матрицы из старой, выкидыванием столбца colon и строки strin

for(int col = 0; col < size; col++){

for(int str = 0; str < size; str++){

if(col != colon && str != strin)

result[str-(str>strin)][col-(col>colon)] = data[str][col];

}

}

resultMinor = det(size-1, result);//Вычисляем минор

for(int i = 0; i < size-1; i++)

delete []result[i];

delete []result;

return resultMinor;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Транспонирование квадратной матрицы

\*\*data - матрица

size - сторона матрицы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void transposeSQR(double \*\*data, int size){

double temp;

for(int str = 0; str < size; str++)

for(int col = str+1; col < size; col++)

data[str][col] =

(temp = data[col][str],

data[col][str] = data[str][col],

temp);

}

# Контрольные примеры работы программы (вычисление обратной матрицы)

Пример 1.

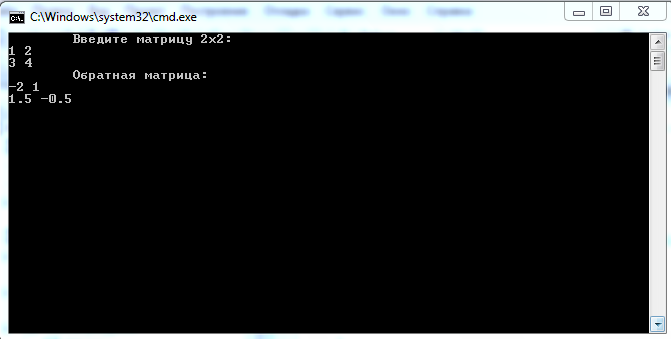


Рис.16. Вычисление обратной матрицы размера 2х2.

Пример 2.

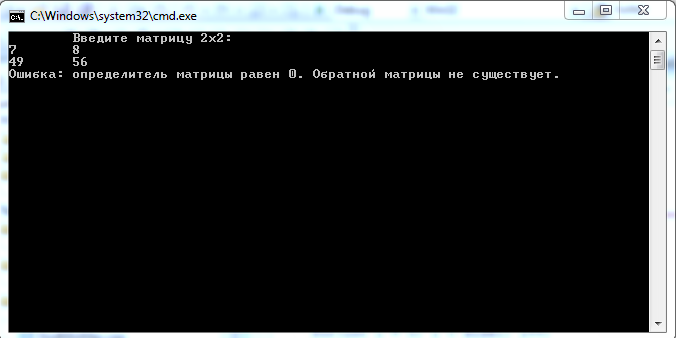


Рис.17. Вычисление обратной матрицы размера 2х2. Программа отреагировала на ошибку.

Пример 3.

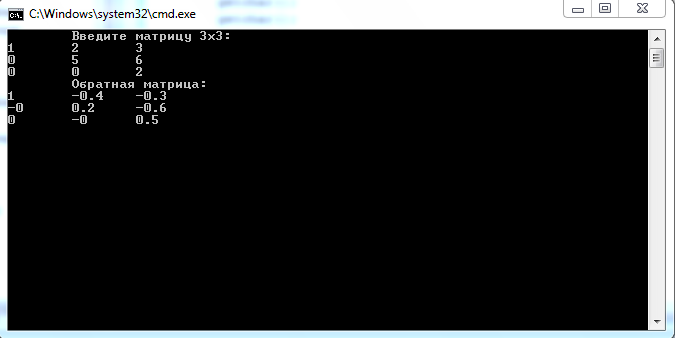


Рис.18. Вычисление обратной матрицы размера 3х3.

Пример 4.

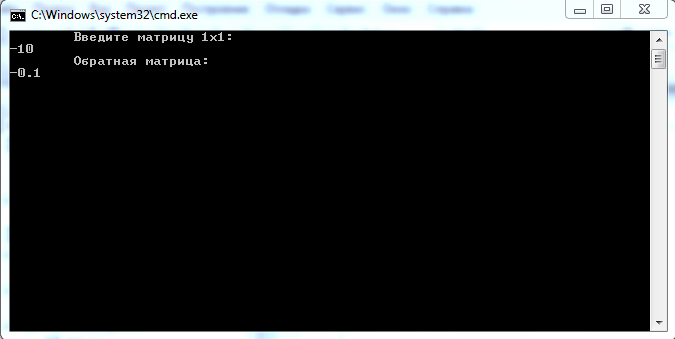


Рис.19. Вычисление обратной матрицы размера 1x1.

Пример 5.

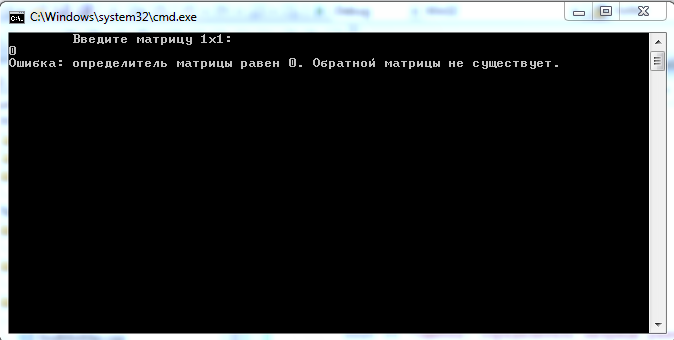


Рис.20. Вычисление обратной матрицы размера 1x1. Программа отреагировала на ошибку.

# Заключение

В ходе данной лабораторной был изучены алгоритмы сортировки (пирамидальная и быстрая сортировка Хоара). Была сделана программная реализация данных алгоритмов. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритмов сортировки.

Была проделана реализация работы с матрицами, а именно: вычисление детерминанта, произведения матриц и нахождение обратной матрицы. Были представлены тестовые примеры, подтверждающие правильность работы программы, а так же разобран ряд критических случаев.