Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнил: студент группы 19ВВ2

Трошкин В.К.

Принял: Д.Т.Н., профессор

Митрохин М.А.

Пенза 2020

**Цель работы:** научиться выполнять основные действия над графами на ЭВМ.

**Ход работы:**

**Задание 1:**

1. Сгенерировали (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Вывели сгенерированные матрицы на экран.

**Задание 2**:

1. Для матричной формы представления графов выполнили операции:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввели с клавиатуры.

Результат выполнения операции вывели на экран.

**Задание 3**

1. Для матричной формы представления графов выполнили операцию:

а) объединения *G* = *G*1  *G*2

б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

Результат выполнения операции вывели на экран.

**Листинг:**

Header.h

#pragma once

char\*\* gen\_adj\_matrix(int size);

int pr\_adj\_matrix(char\*\* matrix, int size);

char\*\* vertex\_cleaving(char\*\* oldmatrix, int size, int to\_cleaving);

char\*\* delete\_vertex(char\*\* oldmatrix, int size, int to\_del);

char\*\* edge\_contraction(char\*\* oldmatrix, int size, char vertex1, char vertex2);

char\*\* vertex\_identification(char\*\* oldmatrix, int size, char vertex1, char vertex2);

char\*\* unite(char\*\* matrix1, char\*\* matrix2, int size);

char\*\* intersect(char\*\* matrix1, char\*\* matrix2, int size);

char\*\* ring\_sum(char\*\* matrix1, char\*\* matrix2, int size);

main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <windows.h>

#include "Header.h"

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int size;

int vertex1, vertex2;

for (;1;) {

printf("1) Удаление вершины\n");

printf("2) Расщепление вершины\n");

printf("3) Стягивание ребра\n");

printf("4) Отождествление вершин\n");

printf("5) Объединение графов\n");

printf("6) Пересечение графов\n");

printf("7) Кольцевая сумма графов\n");

printf("0) Выход\n\n");

switch (\_getch()) {

case ('0'): {

return (0);

}

case ('1'): {

printf("Удаление вершины\n");

printf("Введите размер матрицы: ");

scanf("%d", &size);

char\*\* adj = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj, size);

printf("\nВведите номер вершины для удаления: ");

scanf("%d", &vertex1);

adj = delete\_vertex(adj, size, vertex1);

pr\_adj\_matrix(adj, size-1);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('2'): {

printf("Расщепление вершины\n");

printf("Введите размер матрицы: ");

scanf("%d", &size);

char\*\* adj = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj, size);

printf("\nВведите номер вершины для расщепления: ");

scanf("%d", &vertex1);

adj = vertex\_cleaving(adj, size, vertex1);

pr\_adj\_matrix(adj, size + 1);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('3'): {

printf("Стягивание ребра\n");

printf("Введите размер матрицы: ");

scanf("%d", &size);

char\*\* adj = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj, size);

printf("\nВведите номера вершин для стягивания через пробел: ");

scanf("%d %d", &vertex1, &vertex2);

if (adj = edge\_contraction(adj, size, vertex1, vertex2)) {

pr\_adj\_matrix(adj, size - 1);

}

else

printf("\nМежду вершинами нет ребра");

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('4'): {

printf("Отождествление вершин\n");

printf("Введите размер матрицы: ");

scanf("%d", &size);

char\*\* adj = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj, size);

printf("\nВведите номера вершин для отождествления через пробел: ");

scanf("%d %d", &vertex1, &vertex2);

adj = vertex\_identification(adj, size, vertex1, vertex2);

pr\_adj\_matrix(adj, size - 1);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('5'): {

printf("Объединение графов\n");

printf("Введите размер матриц: ");

scanf("%d", &size);

char\*\* adj1 = gen\_adj\_matrix(size);

Sleep(500);

char\*\* adj2 = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj1, size);

printf("\n");

pr\_adj\_matrix(adj2, size);

printf("\nРезультат\n");

adj1 = unite(adj1, adj2, size);

pr\_adj\_matrix(adj1, size);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('6'): {

printf("Пересечение графов\n");

printf("Введите размер матриц: ");

scanf("%d", &size);

char\*\* adj1 = gen\_adj\_matrix(size);

Sleep(500);

char\*\* adj2 = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj1, size);

printf("\n");

pr\_adj\_matrix(adj2, size);

printf("\nРезультат\n");

adj1 = intersect(adj1, adj2, size);

pr\_adj\_matrix(adj1, size);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

case ('7'): {

printf("Кольцевая сумма графов\n");

printf("Введите размер матриц: ");

scanf("%d", &size);

char\*\* adj1 = gen\_adj\_matrix(size);

Sleep(500);

char\*\* adj2 = gen\_adj\_matrix(size);

pr\_adj\_matrix(adj1, size);

printf("\n");

pr\_adj\_matrix(adj2, size);

printf("\nРезультат\n");

adj1 = ring\_sum(adj1, adj2, size);

pr\_adj\_matrix(adj1, size);

printf("\nНажмите любую клавишу для продолжения");

\_getch();

system("cls");

break;

}

default: printf("Неизвестная команда\n\n");

}

}

}

adjmatrixgen.cpp

#include <time.h>

#include <stdlib.h>

char\*\* gen\_adj\_matrix(int size) {

srand(time(0));

char\*\* matrix = (char\*\*)calloc(size, sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size, sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

for (int j = i; j < size; j++) {

if (i == j) {

matrix[i][j] = 0;

}

else {

matrix[i][j] = rand() % 2;

matrix[j][i] = matrix[i][j];

}

}

}

return matrix;

}

adj\_matr\_unary.cpp

#include <stdlib.h>

#include <string.h>

char\*\* delete\_vertex(char\*\* oldmatrix, int size, int to\_del) {

char\*\* matrix = (char\*\*)calloc(size - long long int(1), sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size - long long int(1), sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

for (int j = i; j < size; j++) {

if ((i != to\_del) && (j != to\_del)) {

matrix[(i < to\_del) ? i : i - 1][(j < to\_del) ? j : j - 1] = oldmatrix[i][j];

matrix[(j < to\_del) ? j : j - 1][(i < to\_del) ? i : i - 1] = matrix[(i < to\_del) ? i : i - 1][(j < to\_del) ? j : j - 1];

}

}

}

return matrix;

}

char\*\* vertex\_cleaving(char\*\* oldmatrix, int size, int to\_cleaving) {

char\*\* matrix = (char\*\*)calloc(size + long long int(1), sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size + long long int(1), sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

for (int j = i; j < size; j++) {

if ((i != to\_cleaving) && (j != to\_cleaving)) {

matrix[i][j] = oldmatrix[i][j];

matrix[j][i] = oldmatrix[i][j];

}

}

}

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

if (i % 2 == 0) {

matrix[to\_cleaving][i] = oldmatrix[to\_cleaving][i];

matrix[i][to\_cleaving] = oldmatrix[to\_cleaving][i];

}

else {

matrix[size][i] = oldmatrix[to\_cleaving][i];

matrix[i][size] = oldmatrix[to\_cleaving][i];

}

}

matrix[size][to\_cleaving] = 1;

matrix[to\_cleaving][size] = 1;

return matrix;

}

char\*\* edge\_contraction(char\*\* oldmatrix, int size, char vertex1, char vertex2) {

if (oldmatrix[vertex1][vertex2] != 1) {

return NULL;

}

if (vertex1 > vertex2) {

char buffer;

buffer = vertex2;

vertex2 = vertex1;

vertex1 = buffer;

}

char\*\* matrix = (char\*\*)calloc(size - long long int(1), sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size - long long int(1), sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

char\* result\_vertex = (char\*)calloc(size - long long int(1), sizeof(char));

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

if (vertex1 || vertex2) {

result\_vertex[(i < vertex2) ? i : i-1] = 1;

}

else

{

result\_vertex[(i < vertex2) ? i : i - 1] = 0;

}

}

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

for (int j = i; j < size; j++) {

if ((i != vertex1) && (j != vertex1) && (i != vertex2) && (j != vertex2)) {

matrix[(i < vertex2) ? i : i - 1][(j < vertex2) ? j : j - 1] = oldmatrix[i][j];

matrix[(j < vertex2) ? j : j - 1][(i < vertex2) ? i : i - 1] = matrix[(i < vertex2) ? i : i - 1][(j < vertex2) ? j : j - 1];

}

}

}

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

if (vertex1 == i) {

matrix[i][vertex1] = 0;

}

else {

matrix[vertex1][i] = result\_vertex[i];

matrix[i][vertex1] = result\_vertex[i];

}

}

return(matrix);

}

char\*\* vertex\_identification(char\*\* oldmatrix, int size, char vertex1, char vertex2) {

if (vertex1 > vertex2) {

char buffer;

buffer = vertex2;

vertex2 = vertex1;

vertex1 = buffer;

}

char\*\* matrix = (char\*\*)calloc(size - long long int(1), sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size - long long int(1), sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

char\* result\_vertex = (char\*)calloc(size - long long int(1), sizeof(char));

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

if (vertex1 || vertex2) {

result\_vertex[(i < vertex2) ? i : i - 1] = 1;

}

else

{

result\_vertex[(i < vertex2) ? i : i - 1] = 0;

}

}

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

for (int j = i; j < size; j++) {

if ((i != vertex1) && (j != vertex1) && (i != vertex2) && (j != vertex2)) {

matrix[(i < vertex2) ? i : i - 1][(j < vertex2) ? j : j - 1] = oldmatrix[i][j];

matrix[(j < vertex2) ? j : j - 1][(i < vertex2) ? i : i - 1] = matrix[(i < vertex2) ? i : i - 1][(j < vertex2) ? j : j - 1];

}

}

}

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

matrix[vertex1][i] = result\_vertex[i];

matrix[i][vertex1] = result\_vertex[i];

}

return(matrix);

}

pr\_output.cpp

#include <stdio.h>

int pr\_adj\_matrix(char\*\* matrix, int size) {

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

return 0;

}

adj\_matrix\_binary.cpp

#include <stdlib.h>

#include <string.h>

char\*\* unite(char\*\* matrix1, char\*\* matrix2, int size) {

char\*\* matrix = (char\*\*)calloc(size, sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size, sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

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

matrix[i][j] = (matrix1[i][j] | matrix2[i][j]);

}

}

return matrix;

}

char\*\* intersect(char\*\* matrix1, char\*\* matrix2, int size) {

char\*\* matrix = (char\*\*)calloc(size, sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size, sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

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

matrix[i][j] = (matrix1[i][j] & matrix2[i][j]);

}

}

return matrix;

}

char\*\* ring\_sum(char\*\* matrix1, char\*\* matrix2, int size) {

char\*\* matrix = (char\*\*)calloc(size, sizeof(char\*));

if (matrix == NULL)

return NULL;

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

matrix[i] = (char\*)calloc(size, sizeof(char));

if (matrix[i] == NULL)

return NULL;

}

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

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

matrix[i][j] = (matrix1[i][j] ^ matrix2[i][j]);

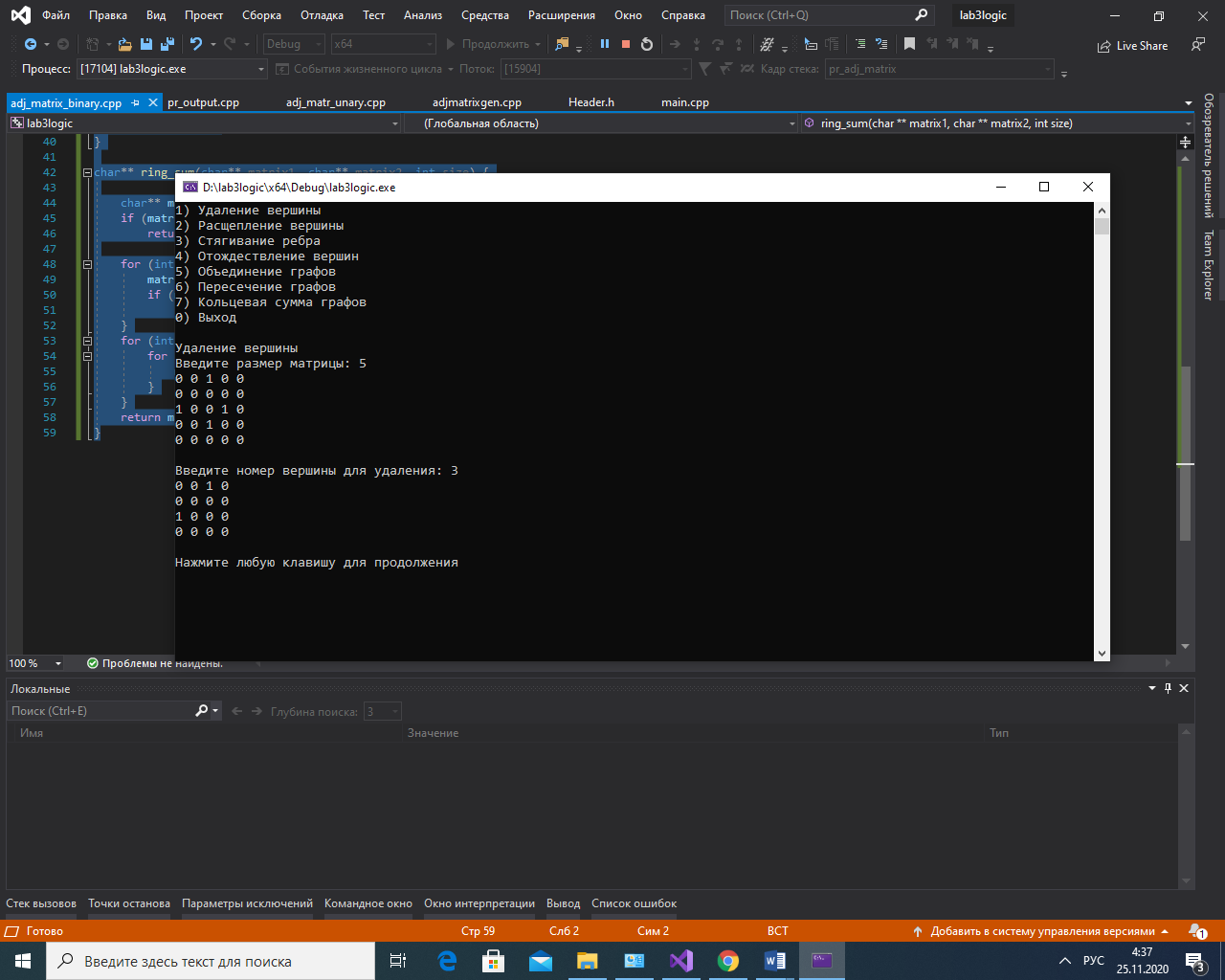
}

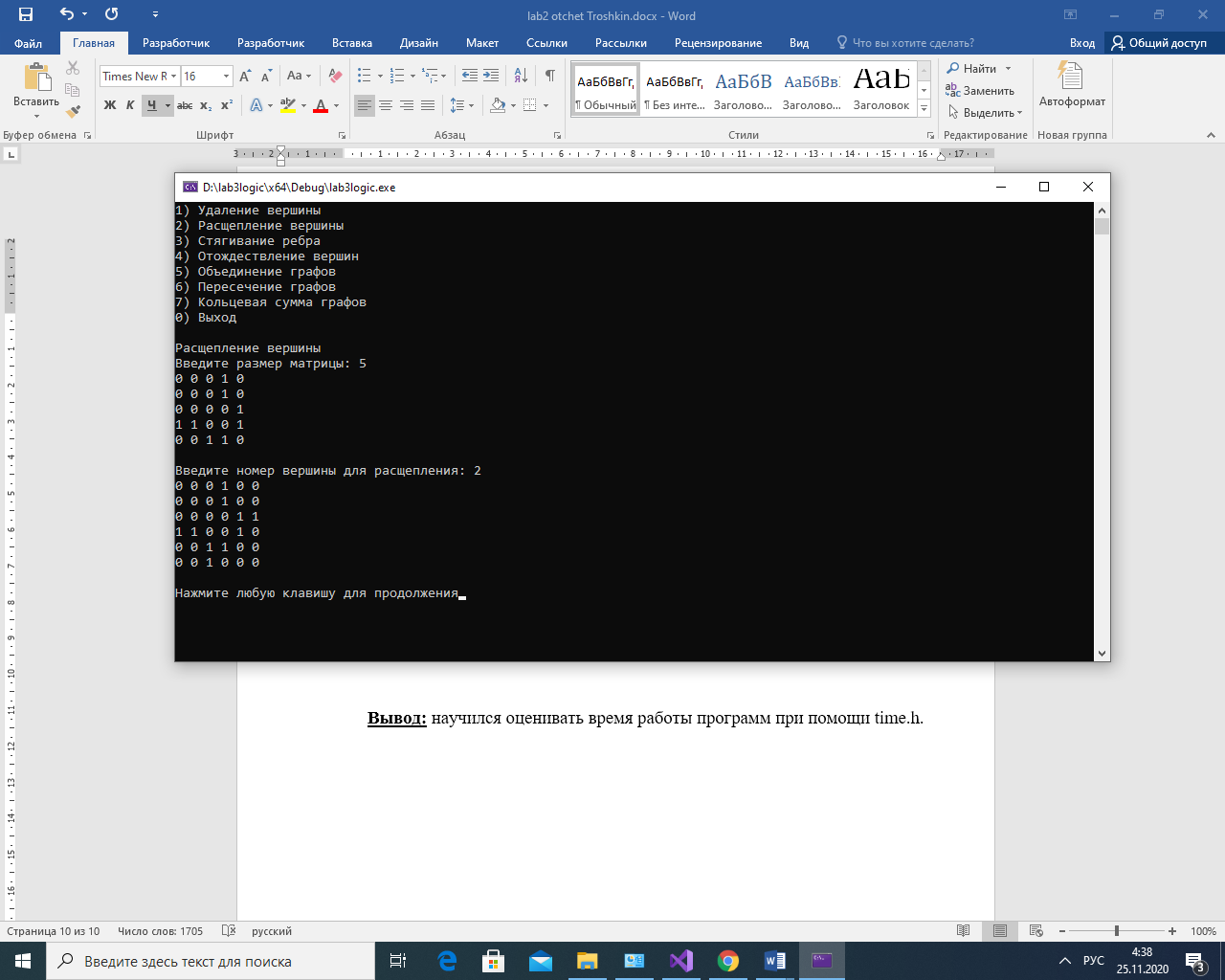
}

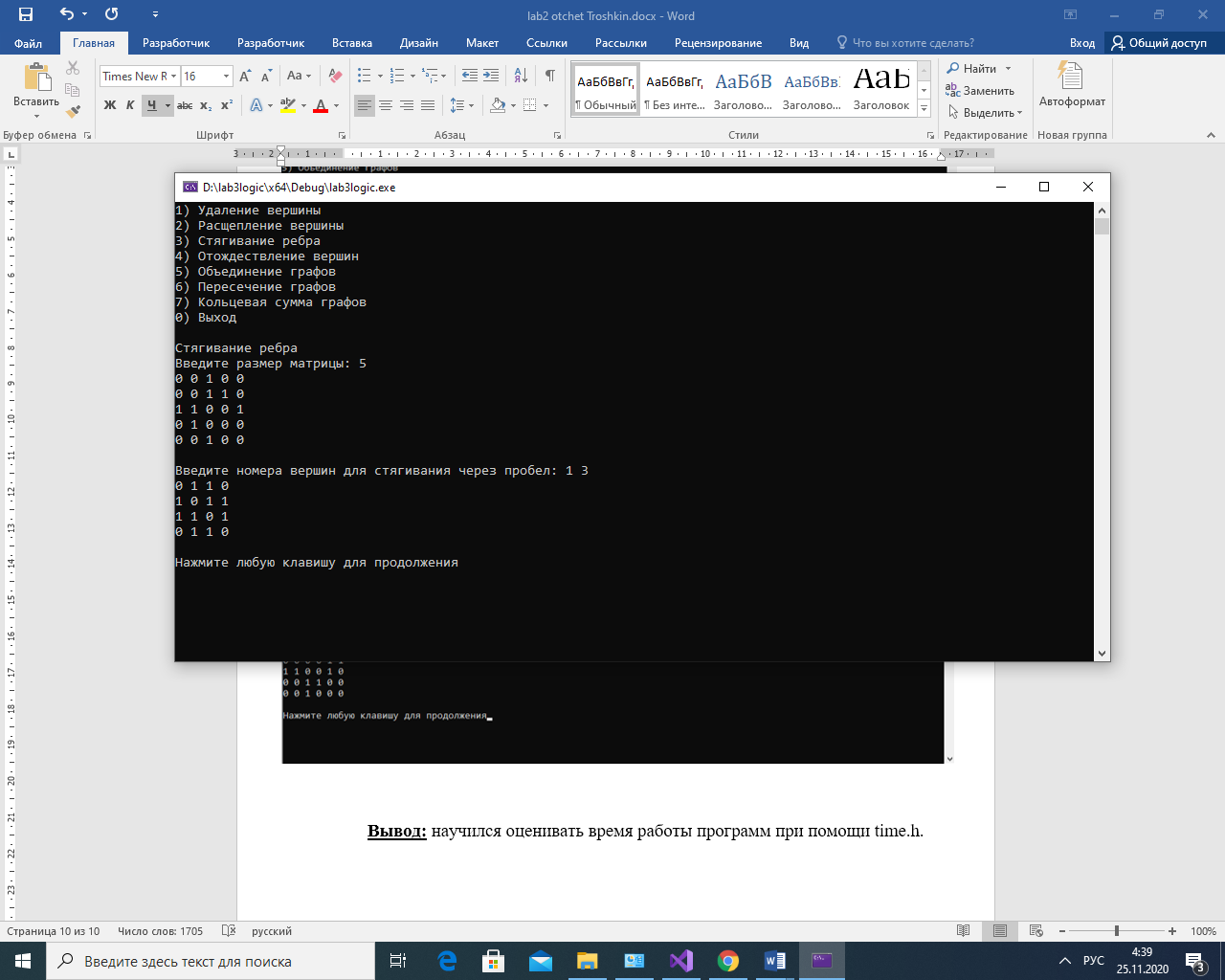
return matrix;

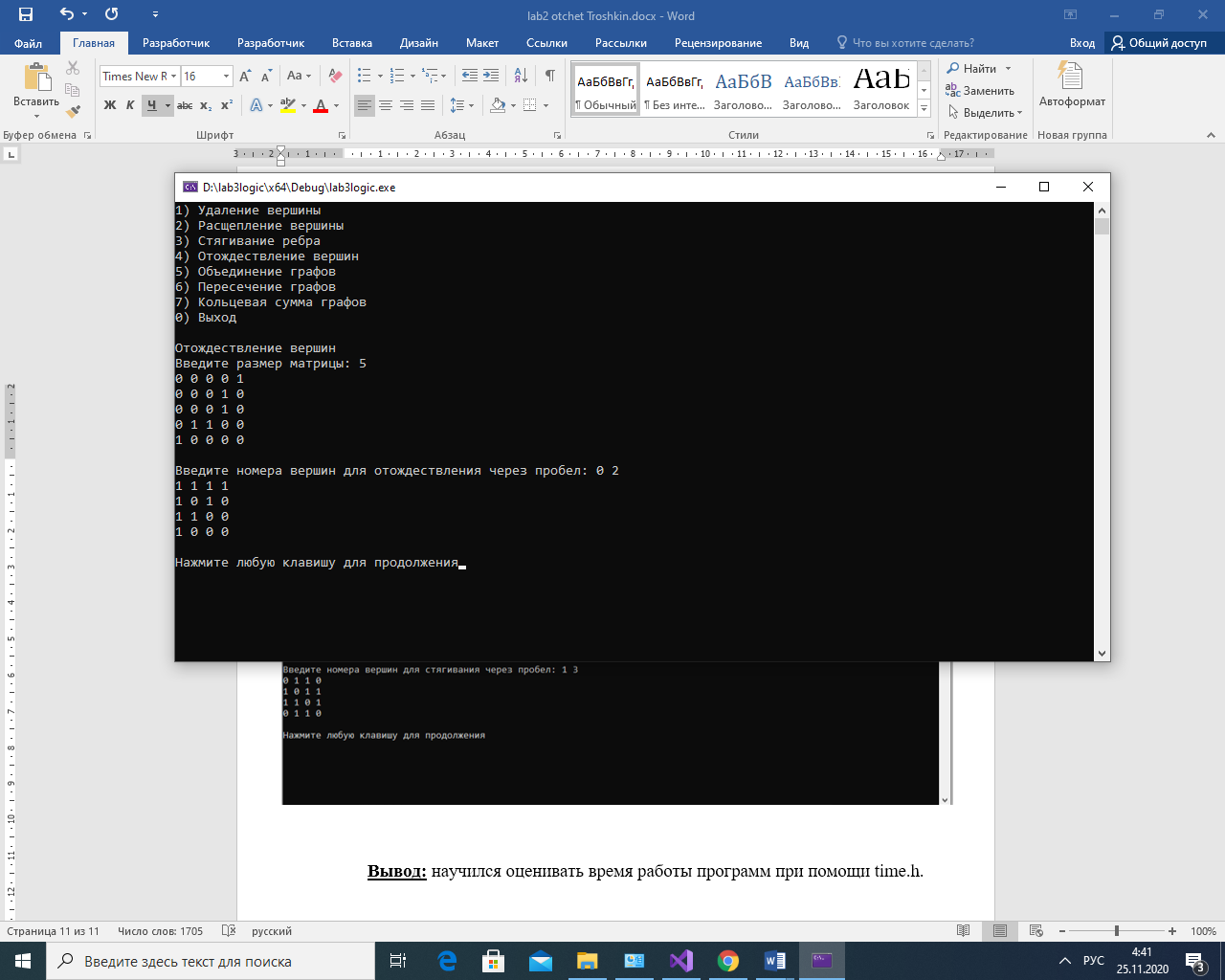
}

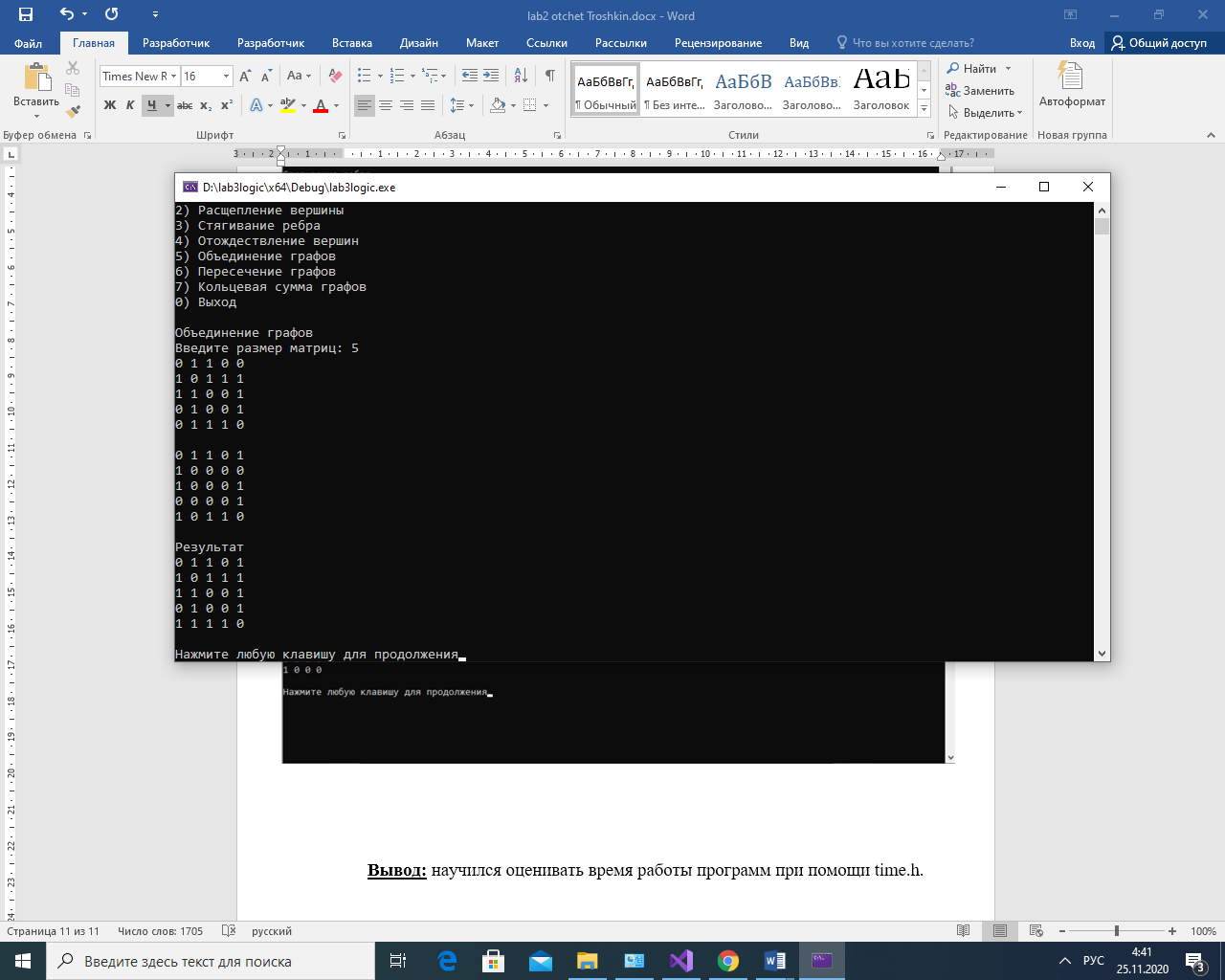
Результаты работы программы:

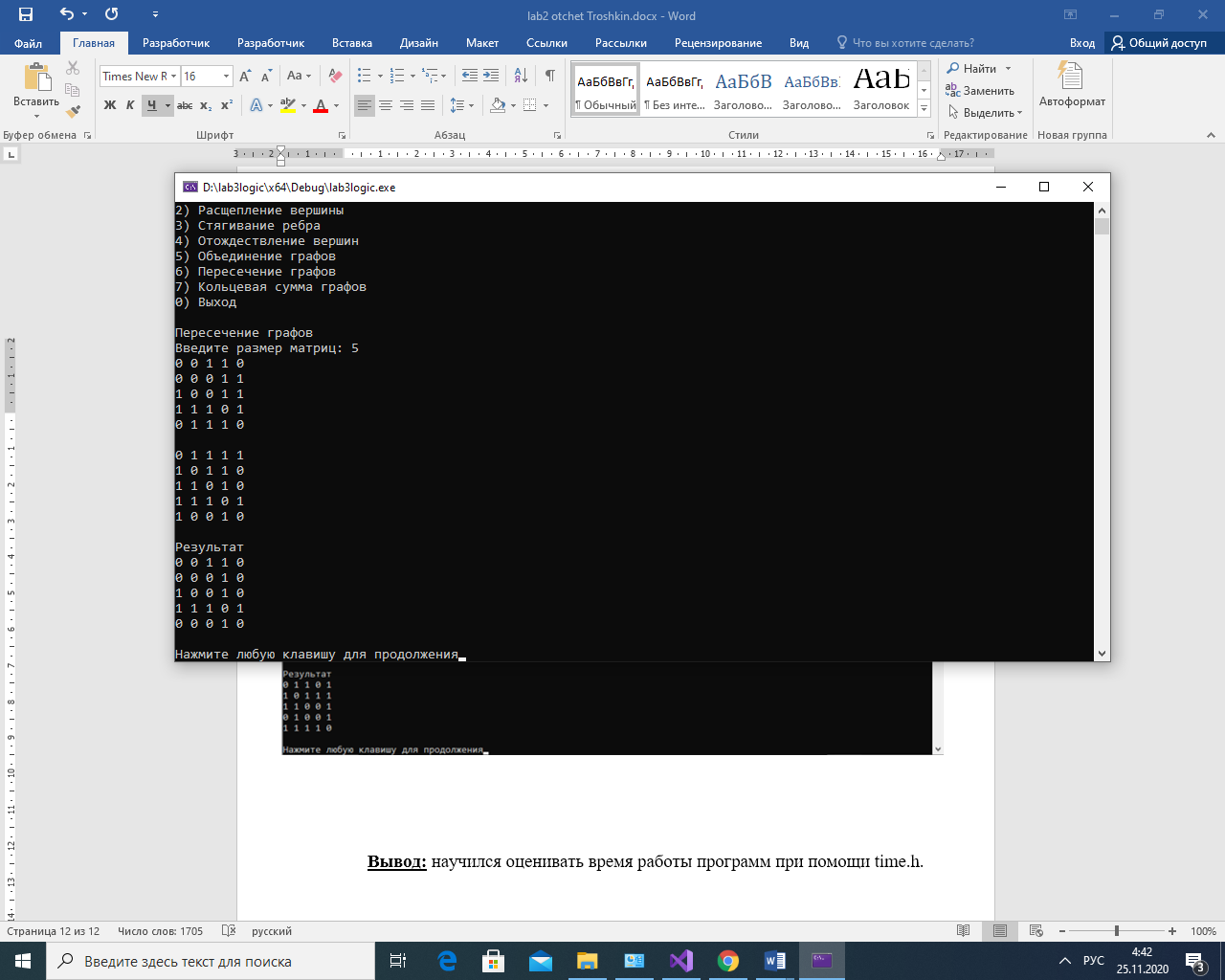


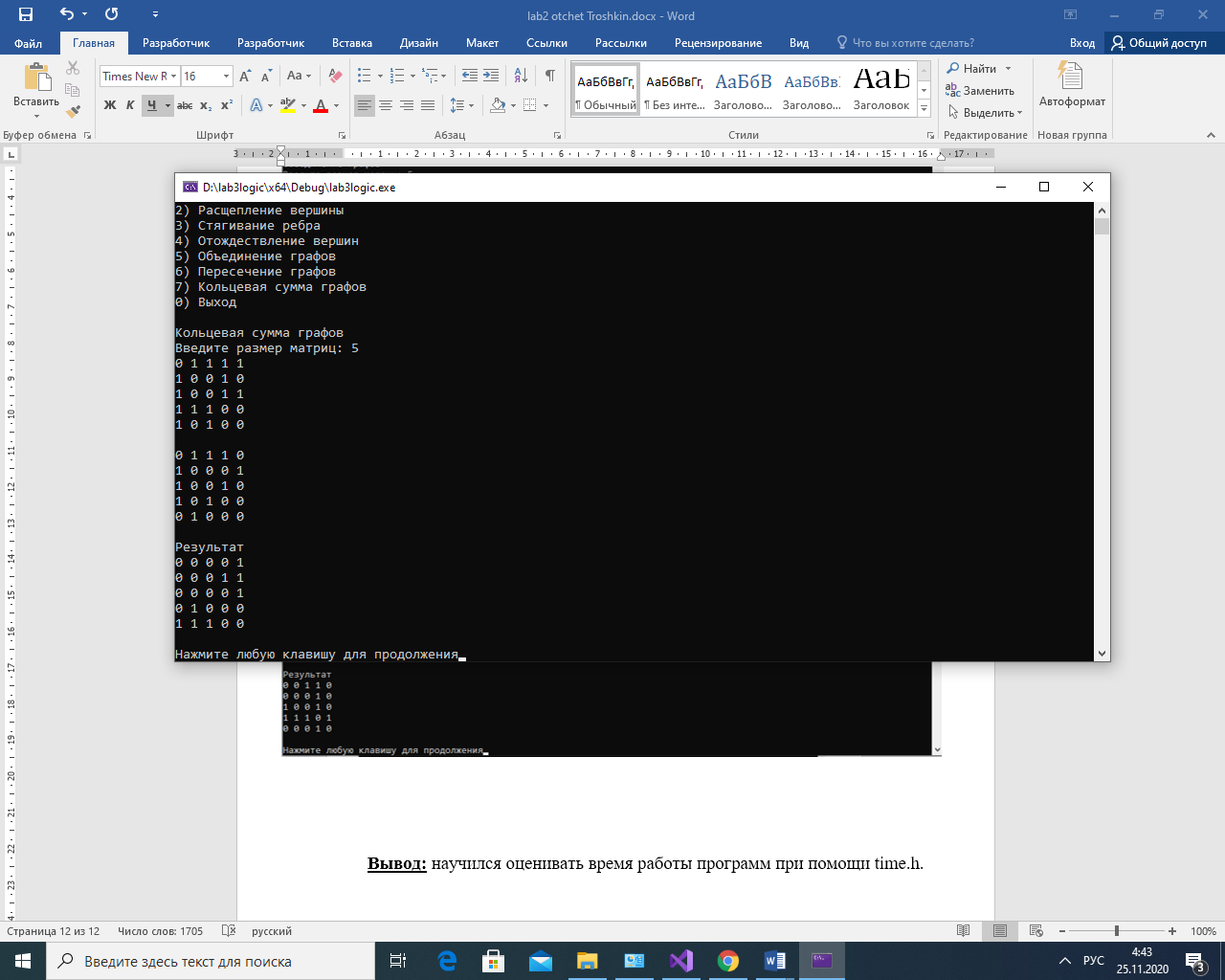












**Вывод:** научился выполнять основные действия над графами на ЭВМ.