Пермский национальный исследовательский  
политехнический университет

Кафедра «Информационные технологии и автоматизированные системы»

**Творческое задание**

Часть1. «Калькулятор решения СЛАУ матричным методом»

Часть 2. «Задача Коммивояжера»

**Отчёт**

Выполнила:

Студент группы ИВТ-21-2б

Азманова Эвелина Робертовна

Проверил: доцент кафедры ИТАС

Полякова О. А.

Пермь, 2022

Часть 1

«Калькулятор решения СЛАУ матричным методом»

Постановка задачи

Необходимо реализовать калькулятор для решения СЛАУ методом обратной матрицы с использованием инструмента Windows Forms. Калькулятор должен выводить поэтапное решение системы (союзную, союзную транспонированную, обратную матрицы, решение СЛАУ). Ввод данных должен осуществляться чтением из файла.

Описание решения

Для решения задачи используется пользовательский тип данных class Matrix, описывающий свойства и методы экземпляров класса. В данной задаче экземпляры класса Matrix – матрицы. Свойствами экземпляров класса являются: размерность (n\*m) (int) и таблица (float\*\*). Методы класса включают: конструкторы (без параметров, с параметрами, копирования), деструктор, селекторы, модификаторы и функции для решения СЛАУ. Для получения корней системы реализованы следующие компонентные функции класса:

1. Функция для нахождения определителя (float Determinant()), которая возвращает значение определителя матрицы. Предусмотрены все способы нахождения определителя (для матрицы размерности (1\*1), размерности (2\*2), размерности (n, m >=3). В случае если порядок определителя матрицы >=3 функция void Minor(Matrix\*, int, int) составляет минор для алгебраического дополнения Аij, далее с помощью алгебраических дополнений высчитывается значение определителя матричной системы.
2. Функция составление союзной матрицы (void union\_Matrix()). Для каждого элемента матричной системы аij находится минор (функция void Minor(Matrix\*, int, int)), затем алгебраическое дополнение. Далее составление союзной матрицы, где на месте аij стоит его алгебраическое дополнение Аij.
3. Функция транспонирования союзной матрицы (void Transposition\_Matrix()). Строки союзной матрицы становятся столбцами транспонированной матрицы, соответственно, столбцы союзной –> строки транспонированной.
4. Функция получения обратной матрицы (void inverse\_Matrix(float)). Для получения обратной матрицы необходимо каждый элемент аij (Аij) транспонированной матрицы целочисленно поделить на определитель.
5. Решение СЛАУ (void solution\_x(Matrix&)). Для нахождения корней системы выполняется перемножение обратной матрицы и столбца свободных коэффициентов. В результате столбец корней размерностью (1\*n).

Все методы класса Matrix для удобства вынесены в отдельный файл Matrix.cpp. В файле реализации самой формы Myform.cpp подключен файл Matrix.cpp, что позволяет обращаться к компонентным функция класса при написании кода для объектов формы. Данные действия дают возможность декомпонировать программу.

Калькулятор реализован при помощи инструмента Windows Forms для удобного использования программы пользователем. Windows Forms позволяют создать интерфейс калькулятора, включающий собственный дизайн, функциональные кнопки, окна ввода/вывода информации.

Дизайн калькулятора выполнен в голубых, серых оттенках, так как человеку комфортнее работать с интерфейсами в пастельных тонах. Калькулятор выводит все этапы решения СЛАУ, поэтому дизайн содержит несколько серых подложек для ввода/вывода матриц, при этом очередность этапов решения соблюдена. Интерфейс достаточно прост, понятен и удобен в использовании (рис.1).

Основное средство визуализации матриц – элемент dataGridView, который является надстраиваемой таблицей, что очень удобно для отображения матричных систем. Данный элемент управления имеет огромное число надстроек, благодаря этому возможно построить любую таблицу. Пример вывода данных с использованием dataGridView (рис.2).

В ходе работы были трудности в использовании элемента dataGridView, а именно с настройкой размеров строк и столбцов таблицы. Проблема решена данными встроенными функциями: AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders); AutoResizeColumns().

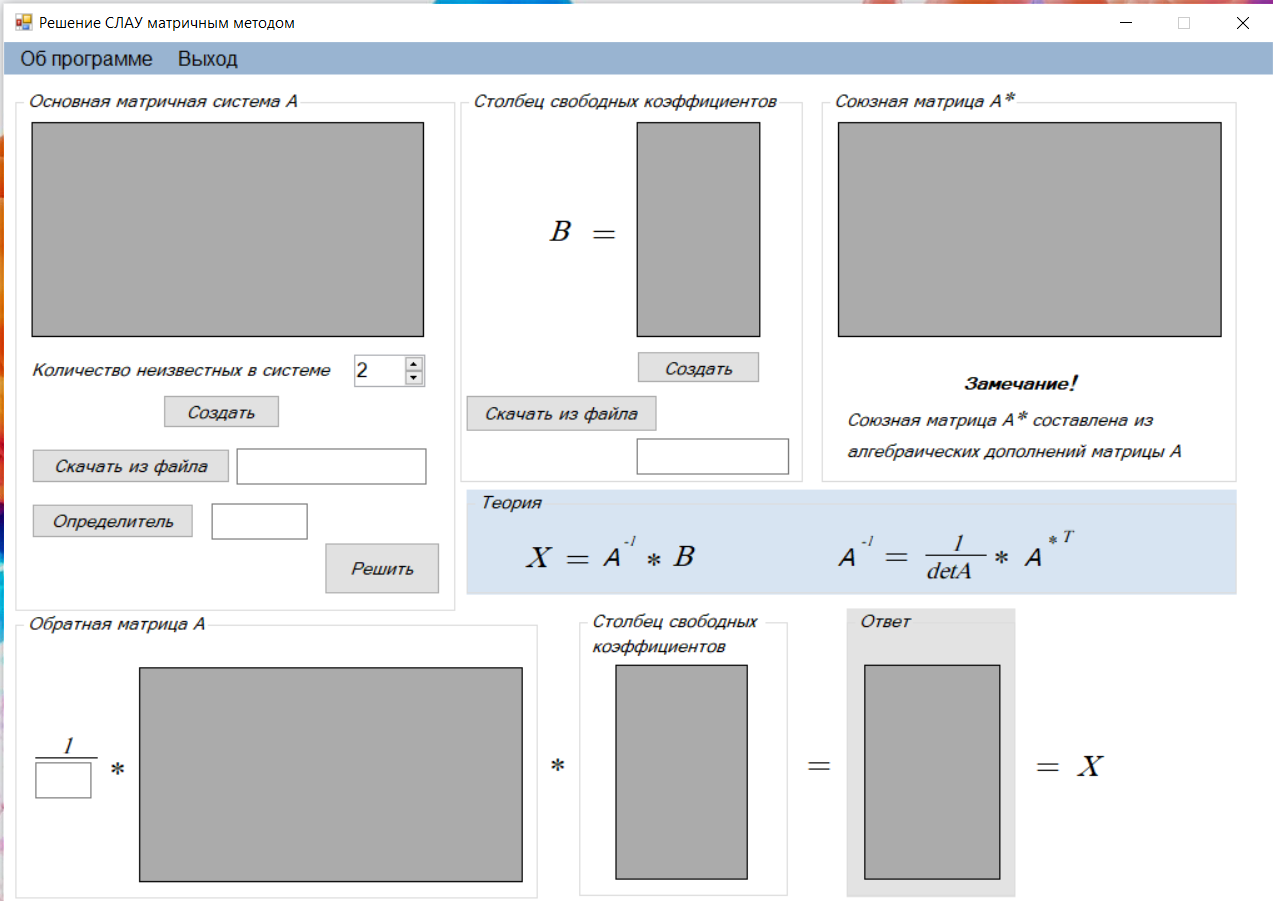


Рис. 1 «Интерфейс калькулятора»

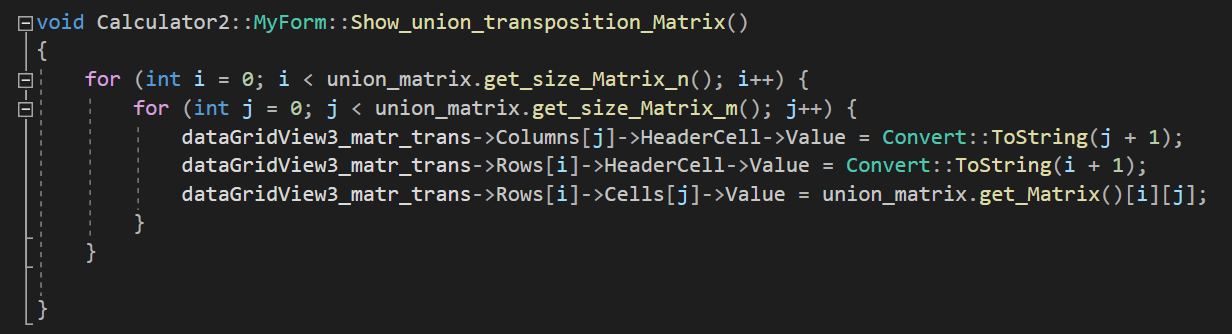


Рис. 2 «Использование элемента dataGridView» для »

Особенностями и достоинствами данной программы являются:

1. Удобный интерфейс, не нагруженный излишними функциями.
2. Вывод этапов решения СЛАУ.
3. Ввод данных чтением из файла, что позволяет сократить время на заполнение значениями основной матричной системы.
4. Имеются краткие теоретические сведения по решению СЛАУ.
5. Предусмотрена кнопка выхода из калькулятора.
6. Предусмотрен вывод сообщений пользователю:

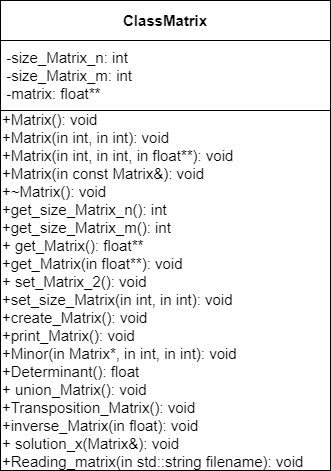
а) справка об программе

б) Ошибка при определителе равном 0

в) Ошибка чтения файла

1. Предусмотрено, что матричная система должна быть квадратной, поэтому пользователь определяет одной кнопкой таблицу (n\*n) и количество неизвестных в системе не может быть меньше 2.

Диаграмма классов



Код программы

#pragma once

#include <iostream>

#include<random>

#include <cmath>

#include<fstream>

using namespace System;

std::string& Convert\_String\_to\_string(String^ s, std::string& os); // Конвертируем System:: srting^ в std::string

String^ Convert\_string\_to\_String(std::string& os, String^ s); // Конвертируем std::string в System:: srting^

#pragma once

#include "Functions.h"

class Matrix

{

private:

int size\_Matrix\_n;

int size\_Matrix\_m;

float\*\* matrix;

public:

Matrix();

Matrix(int, int);

Matrix(int, int, float\*\*);

Matrix(const Matrix&);

~Matrix();

int get\_size\_Matrix\_n();

int get\_size\_Matrix\_m();

float\*\* get\_Matrix();

void set\_Matrix(float\*\*);

void set\_Matrix\_2();

void set\_size\_Matrix(int,int);

void create\_Matrix();

void print\_Matrix();

void Minor(Matrix\*, int, int);

float Determinant();

void union\_Matrix();

void Transposition\_Matrix();

void inverse\_Matrix(float);

void solution\_x(Matrix&);

void Reading\_matrix(std::string filename);

};

#include"Matrix.h"

using namespace System;

using namespace System::Windows::Forms;

Matrix::Matrix() {

size\_Matrix\_n = 0;

size\_Matrix\_m = 0;

matrix = new float\* [size\_Matrix\_n];

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

matrix[i] = new float[size\_Matrix\_m];

}

}

Matrix::Matrix(int size\_matrix\_N, int size\_matrix\_M) {

size\_Matrix\_n = size\_matrix\_N;

size\_Matrix\_m = size\_matrix\_M;

matrix = new float\* [size\_Matrix\_n];

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

matrix[i] = new float[size\_Matrix\_m];

}

}

Matrix::Matrix(int size\_matrix\_N, int size\_matrix\_M, float\*\* Matrix) {

size\_Matrix\_n = size\_matrix\_N;

size\_Matrix\_m = size\_matrix\_M;

matrix = new float\* [size\_Matrix\_n];

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

matrix[i] = new float[size\_Matrix\_m];

}

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

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

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

}

}

}

Matrix::Matrix(const Matrix& to\_copy) {

this->size\_Matrix\_n = to\_copy.size\_Matrix\_n;

this->size\_Matrix\_m = to\_copy.size\_Matrix\_m;

this->matrix = to\_copy.matrix;

}

Matrix::~Matrix() {

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

delete[] matrix[i];

}

delete[]matrix;

}

int Matrix::get\_size\_Matrix\_n() {

return size\_Matrix\_n;

}

int Matrix::get\_size\_Matrix\_m() {

return size\_Matrix\_m;

}

float\*\* Matrix::get\_Matrix() {

return matrix;

}

void Matrix::set\_Matrix(float\*\* Matrix) {

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

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

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

}

}

}

void Matrix::set\_Matrix\_2() {

for (int i = 0; i < this->size\_Matrix\_n; i++) {

for (int j = 0; j < this->size\_Matrix\_m; j++) {

//cin >> this->matrix[i][j];

}

}

}

void Matrix::set\_size\_Matrix(int size\_matrix\_N, int size\_matrix\_M) {

size\_Matrix\_n = size\_matrix\_N;

size\_Matrix\_m = size\_matrix\_M;

matrix = new float\* [size\_Matrix\_n];

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

matrix[i] = new float[size\_Matrix\_m];

}

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

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

matrix[i][j] = 0;

}

}

}

void Matrix::create\_Matrix() {

srand(time(NULL));

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

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

matrix[i][j] = rand() % 10;

}

}

}

void Matrix::print\_Matrix() {

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

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

// cout << matrix[i][j] << " ";

}

//cout << endl;

}

}

void Matrix::Minor(Matrix\* temp\_matrix, int indRow, int indCol) { // получение минора матрицы

int ki = 0;

for (int i = 0; i < this->size\_Matrix\_n; i++) {

if (i != indRow) {

for (int j = 0, kj = 0; j < this->size\_Matrix\_m; j++) {

if (j != indCol) {

temp\_matrix->matrix[ki][kj] = this->matrix[i][j];

kj++;

}

}

ki++;

}

}

}

float Matrix::Determinant() { // определитель матрицы

float temp = 0;

int k = 1;

if (this->size\_Matrix\_n < 1) {

MessageBox::Show("Не верный размер матрицы!", "Ошибка!");

return 0;

}

else if (this->size\_Matrix\_n == 1)

temp = matrix[0][0];

else if (this->size\_Matrix\_n == 2)

temp = this->matrix[0][0] \* this->matrix[1][1] - this->matrix[1][0] \* this->matrix[0][1];

else {

for (int i = 0; i < this->size\_Matrix\_n; i++) {

int m = this->size\_Matrix\_n - 1;

Matrix temp\_matrix(m, m);

this->Minor(&temp\_matrix, 0, i);

temp = temp + k \* matrix[0][i] \* temp\_matrix.Determinant();

k = -k;

}

}

return temp;

}

void Matrix::union\_Matrix() { // создание союзной матрицы из алгебраических дополнений каждого элемента

Matrix union\_matrix(this->size\_Matrix\_n, this->size\_Matrix\_m);

float temp = 0;

for (int i = 0; i < this->size\_Matrix\_n; i++) {

for (int j = 0, kj = 0; j < this->size\_Matrix\_m; j++) {

int m = this->size\_Matrix\_n - 1;

Matrix temp\_matrix(m, m);

this->Minor(&temp\_matrix, i, j);

temp = temp\_matrix.Determinant() \* pow(-1, i + j);

union\_matrix.matrix[i][j] = temp;

}

}

this->set\_Matrix(union\_matrix.matrix);

}

void Matrix::Transposition\_Matrix() { // транспонирование союзной матрицы

Matrix transposition\_matrix(this->size\_Matrix\_n, this->size\_Matrix\_m);

for (int i = 0; i < this->size\_Matrix\_n; i++) {

for (int j = 0; j < this->size\_Matrix\_m; j++) {

transposition\_matrix.matrix[j][i] = this->matrix[i][j];

}

}

this->set\_Matrix(transposition\_matrix.matrix);

}

void Matrix::inverse\_Matrix(float det\_matrix) { // получение обратной матрицы

float det = det\_matrix;

Matrix inverse\_matrix(this->size\_Matrix\_n, this->size\_Matrix\_m);

for (int i = 0; i < this->size\_Matrix\_n; i++) {

for (int j = 0; j < this->size\_Matrix\_n; j++) {

inverse\_matrix.matrix[i][j] = (this->matrix[i][j] / det);

}

}

this->set\_Matrix(inverse\_matrix.matrix);

}

void Matrix::solution\_x(Matrix& x) { // умножение обратной матрицы на столбец свободных коэффициентов

float result = 0;

Matrix solution\_matr(this->size\_Matrix\_n, x.size\_Matrix\_m);

for (int i = 0; i < this->size\_Matrix\_n; i++) {

for (int j = 0; j < this->size\_Matrix\_m; j++) {

result += this->matrix[i][j] \* x.matrix[j][0];

}

solution\_matr.matrix[i][0] = result;

result = 0;

}

x.set\_Matrix(solution\_matr.matrix);

}

void Matrix::Reading\_matrix(std::string filename) { // чтение данных из файлов

std::ifstream reading(filename); // поток для чтения из файла

if (reading) { // если файл открылся

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

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

reading >> matrix[i][j];

}

}

}

else {

MessageBox::Show("Ошибка открытия файла!", "Внимание!");

}

reading.close();

}

#include "Functions.h"

std::string& Convert\_String\_to\_string(String^ s, std::string& os)

{

using namespace Runtime::InteropServices;

const char\* chars = (const char\*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();

os = chars;

Marshal::FreeHGlobal(IntPtr((void\*)chars));

return os;

}

System::String^ Convert\_string\_to\_String(std::string& os, String^ s)

{

s = gcnew System::String(os.c\_str());

return s;

}

#include "MyForm.h"

#include"Matrix.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

void main(array <String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Calculator2::MyForm form;

Application::Run(% form);

}

float det\_matrix\_1;

int matrix\_size\_n, matrix\_size\_m;

Matrix matrix\_1, matrix\_b,union\_matrix, transposition\_matrix,inverse\_matrix, matrix\_x;

System::Void Calculator2::MyForm::обПрограммеToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

MessageBox::Show("Калькулятор находит решение системы n линейных уравнений с n неизвестными методом обратной матрицы ", "Справка");

return System::Void();

}

System::Void Calculator2::MyForm::выходToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

Application::Exit();

return System::Void();

}

System::Void Calculator2::MyForm::button1\_set\_matr\_Click(System::Object^ sender, System::EventArgs^ e)

{

matrix\_size\_n = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_size\_m = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_1.set\_size\_Matrix(matrix\_size\_n, matrix\_size\_m);

dataGridView1\_matr\_1->RowCount = matrix\_size\_n;

dataGridView1\_matr\_1->ColumnCount = matrix\_size\_m;

Show\_Matrix\_1();

dataGridView1\_matr\_1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1\_matr\_1->AutoResizeColumns();

return System::Void();

}

System::Void Calculator2::MyForm::button1\_solution\_Click(System::Object^ sender, System::EventArgs^ e)

{

union\_matrix = matrix\_1;

matrix\_size\_n = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_size\_m = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

dataGridView3\_matr\_trans->RowCount = matrix\_size\_n;

dataGridView3\_matr\_trans->ColumnCount = matrix\_size\_m;

union\_matrix.union\_Matrix();

Show\_union\_transposition\_Matrix();

dataGridView3\_matr\_trans->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView3\_matr\_trans->AutoResizeColumns();

dataGridView1\_inverse\_matr->RowCount = matrix\_size\_n;

dataGridView1\_inverse\_matr->ColumnCount = matrix\_size\_m;

textBox1\_det\_for\_inverse->Text = Convert::ToString(det\_matrix\_1);

transposition\_matrix = union\_matrix;

transposition\_matrix.Transposition\_Matrix();

Show\_invers\_Matrix();

dataGridView1\_inverse\_matr->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1\_inverse\_matr->AutoResizeColumns();

dataGridView1\_b\_for\_solution->RowCount = matrix\_size\_n;

dataGridView1\_b\_for\_solution->ColumnCount = 1;

Show\_Matrix\_b\_for\_solution();

dataGridView1\_b\_for\_solution->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1\_b\_for\_solution->AutoResizeColumns();

inverse\_matrix = transposition\_matrix;

inverse\_matrix.inverse\_Matrix(det\_matrix\_1);

matrix\_x = matrix\_b;

inverse\_matrix.solution\_x(matrix\_x);

dataGridView1\_solution\_x->RowCount = matrix\_size\_n;

dataGridView1\_solution\_x->ColumnCount = 1;

Show\_Matrix\_for\_solution();

dataGridView1\_solution\_x->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1\_solution\_x->AutoResizeColumns();

return System::Void();

}

System::Void Calculator2::MyForm::button1\_set\_b\_Click(System::Object^ sender, System::EventArgs^ e)

{

matrix\_size\_n = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_size\_m = 1;

matrix\_b.set\_size\_Matrix(matrix\_size\_n, matrix\_size\_m);

dataGridView2\_matr\_b->RowCount = matrix\_size\_n;

dataGridView2\_matr\_b->ColumnCount = matrix\_size\_m;

Show\_Matrix\_b();

dataGridView2\_matr\_b->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView2\_matr\_b->AutoResizeColumns();

return System::Void();

}

System::Void Calculator2::MyForm::button1\_det\_Click(System::Object^ sender, System::EventArgs^ e)

{

det\_matrix\_1 = matrix\_1.Determinant();

textBox1\_det->Text = Convert::ToString(det\_matrix\_1);

if (det\_matrix\_1 == 0) {

MessageBox::Show("Определитель не может быть равен 0!", "Внимание!");

}

return System::Void();

}

System::Void Calculator2::MyForm::button1\_solution\_inverse\_b\_Click(System::Object^ sender, System::EventArgs^ e)

{

return System::Void();

}

System::Void Calculator2::MyForm::button1\_file\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (textBox1\_file->Text == " ") {

MessageBox::Show("Введите название файла!", "Внимание!");

return;

}

std::string filename;

Convert\_String\_to\_string(textBox1\_file->Text, filename);

matrix\_size\_n = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_size\_m = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_1.set\_size\_Matrix(matrix\_size\_n, matrix\_size\_n);

dataGridView1\_matr\_1->RowCount = matrix\_size\_n;

dataGridView1\_matr\_1->ColumnCount = matrix\_size\_m;

matrix\_1.Reading\_matrix(filename);

Show\_Matrix\_1();

dataGridView1\_matr\_1->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView1\_matr\_1->AutoResizeColumns();

return System::Void();

}

System::Void Calculator2::MyForm::button1\_file\_b\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (textBox1\_b->Text == " ") {

MessageBox::Show("Введите название файла!", "Внимание!");

return;

}

std::string filename;

Convert\_String\_to\_string(textBox1\_b->Text, filename);

matrix\_size\_n = Convert::ToInt32(numericUpDown1\_size\_n\_m->Value);

matrix\_size\_m = 1;

matrix\_b.set\_size\_Matrix(matrix\_size\_n, matrix\_size\_m);

dataGridView2\_matr\_b->RowCount = matrix\_size\_n;

dataGridView2\_matr\_b->ColumnCount = matrix\_size\_m;

matrix\_b.Reading\_matrix(filename);

Show\_Matrix\_b();

dataGridView2\_matr\_b->AutoResizeRowHeadersWidth(DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);

dataGridView2\_matr\_b->AutoResizeColumns();

return System::Void();

}

void Calculator2::MyForm::Show\_Matrix\_1() {

for (int i = 0; i < matrix\_1.get\_size\_Matrix\_n(); i++) {

for (int j = 0; j < matrix\_1.get\_size\_Matrix\_m(); j++) {

dataGridView1\_matr\_1->Columns[j]->HeaderCell->Value = Convert::ToString(j + 1);

dataGridView1\_matr\_1->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView1\_matr\_1->Rows[i]->Cells[j]->Value = matrix\_1.get\_Matrix()[i][j];

}

}

}

void Calculator2::MyForm::Show\_Matrix\_b()

{

for (int i = 0; i < matrix\_b.get\_size\_Matrix\_n(); i++) {

dataGridView2\_matr\_b->Columns[0]->HeaderCell->Value = Convert::ToString(1);

dataGridView2\_matr\_b->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView2\_matr\_b->Rows[i]->Cells[0]->Value = matrix\_b.get\_Matrix()[i][0];

}

}

void Calculator2::MyForm::Show\_union\_transposition\_Matrix()

{

for (int i = 0; i < union\_matrix.get\_size\_Matrix\_n(); i++) {

for (int j = 0; j < union\_matrix.get\_size\_Matrix\_m(); j++) {

dataGridView3\_matr\_trans->Columns[j]->HeaderCell->Value = Convert::ToString(j + 1);

dataGridView3\_matr\_trans->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView3\_matr\_trans->Rows[i]->Cells[j]->Value = union\_matrix.get\_Matrix()[i][j];

}

}

}

void Calculator2::MyForm::Show\_invers\_Matrix()

{

for (int i = 0; i < transposition\_matrix.get\_size\_Matrix\_n(); i++) {

for (int j = 0; j < transposition\_matrix.get\_size\_Matrix\_m(); j++) {

dataGridView1\_inverse\_matr->Columns[j]->HeaderCell->Value = Convert::ToString(j + 1);

dataGridView1\_inverse\_matr->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView1\_inverse\_matr->Rows[i]->Cells[j]->Value = transposition\_matrix.get\_Matrix()[i][j];

}

}

}

void Calculator2::MyForm::Show\_Matrix\_b\_for\_solution() {

for (int i = 0; i < matrix\_b.get\_size\_Matrix\_n(); i++) {

dataGridView1\_b\_for\_solution->Columns[0]->HeaderCell->Value = Convert::ToString(1);

dataGridView1\_b\_for\_solution->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView1\_b\_for\_solution->Rows[i]->Cells[0]->Value = matrix\_b.get\_Matrix()[i][0];

}

}

void Calculator2::MyForm::Show\_Matrix\_for\_solution()

{

for (int i = 0; i < matrix\_x.get\_size\_Matrix\_n(); i++) {

dataGridView1\_solution\_x->Columns[0]->HeaderCell->Value = Convert::ToString(1);

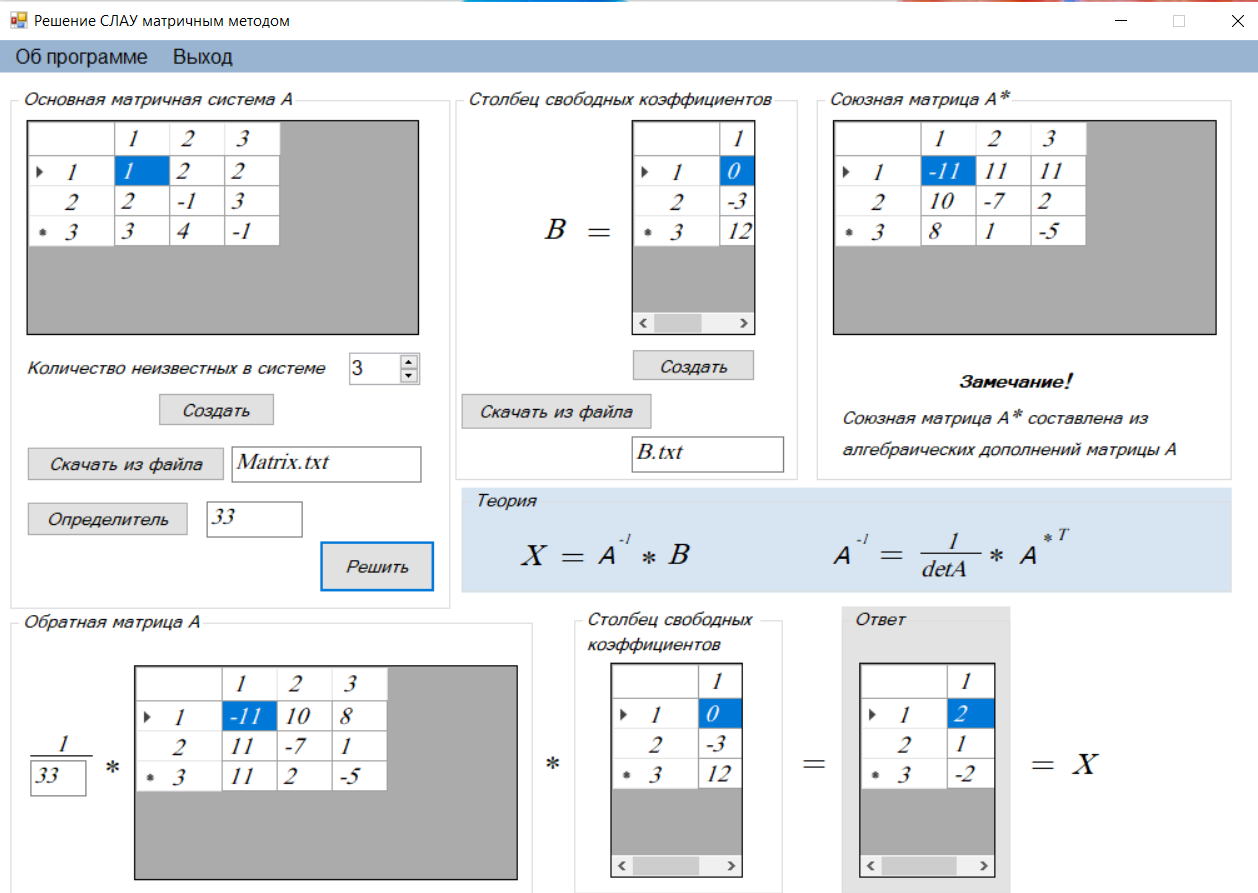
dataGridView1\_solution\_x->Rows[i]->HeaderCell->Value = Convert::ToString(i + 1);

dataGridView1\_solution\_x->Rows[i]->Cells[0]->Value = matrix\_x.get\_Matrix()[i][0];

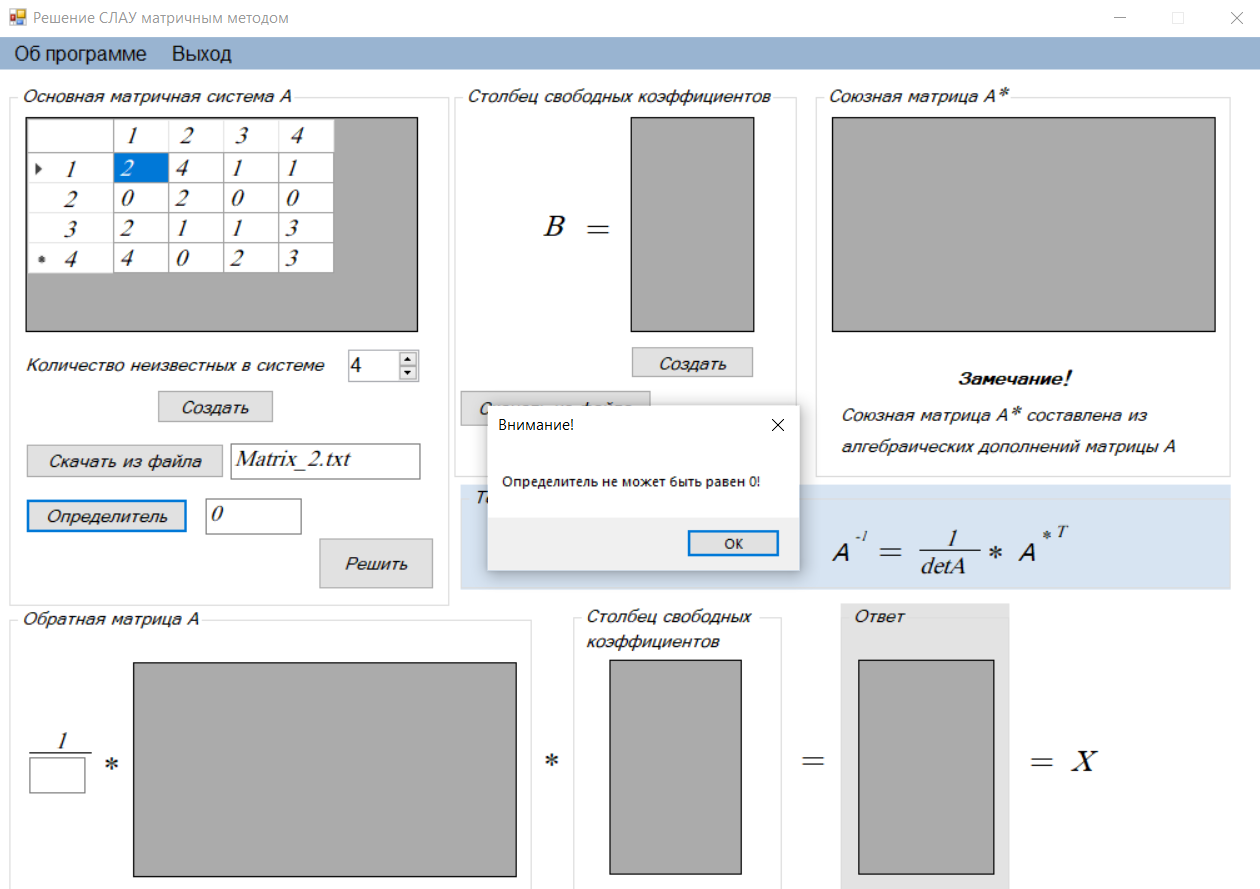
}

}

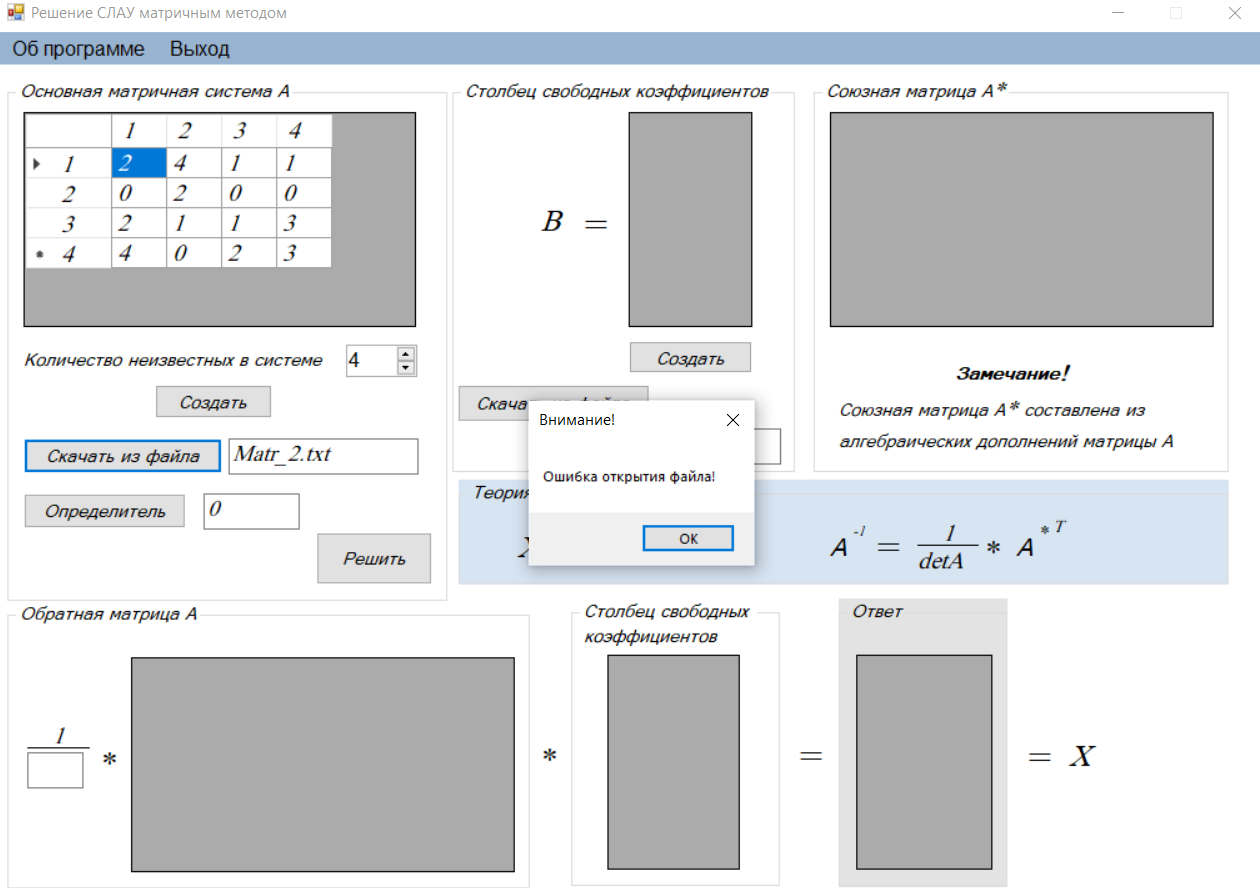
Скриншоты работы

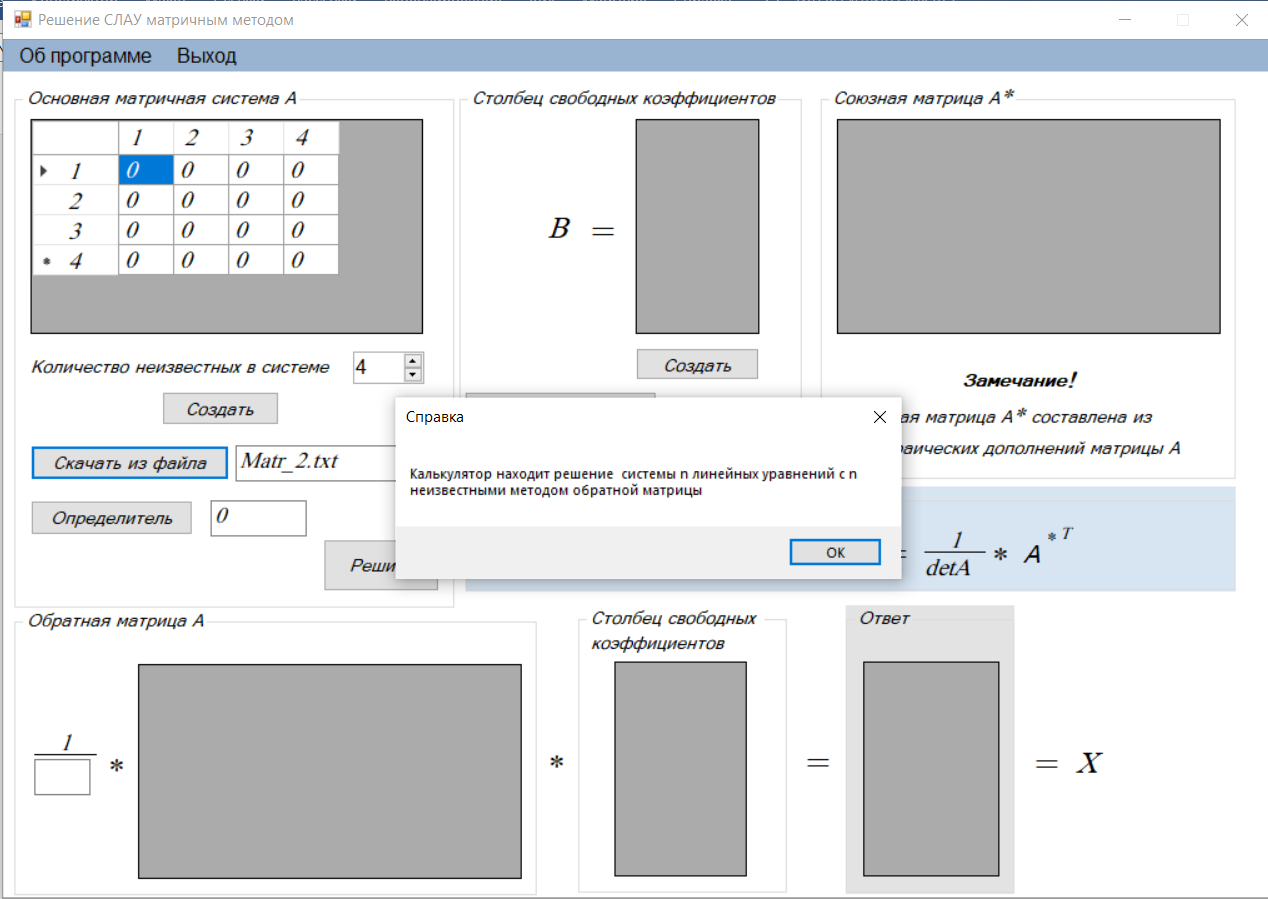


* Определитель не равен нулю



* Ошибка чтения файла



* Справка программы

Часть 2

«Задача Коммивояжера»

Постановка задачи

Коммивояжер должен посетить однократно все города, связанные дорожной сетью, и вернуться в исходный город. Время пути в часах от одного города в другой различно. По условию задачи Коммивояжеру необходимо посетить города, затратив наименьшее количество часов на дорогу.

Дополнительные возможности передвижения из города в город:

1. На автомобиле из города A в город B – время пути сокращается на 3 часа.
2. На вертолете из города A в город B – время пути сокращается на 4 часа.
3. На электросамокате из города A в город B – время пути сокращается на 1 час.

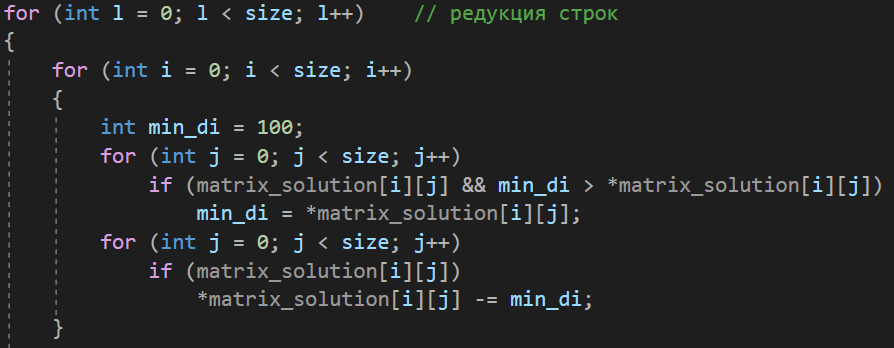
Реализовать граф, где вершины графа– города, а ребра графа – дороги между городами. Предусмотреть возможности добавления новой вершины/ребра, создания нового графа, очистки исходного графа. Визуализацию задачи выполнить при помощи инструмента OpenGL.

Описание решения

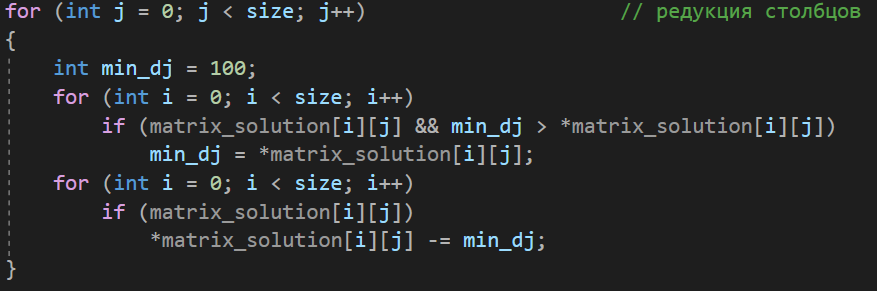
Для решения задачи используется пользовательский тип данных class Graph, описывающий свойства и методы экземпляров класса. В данной задаче экземпляры класса Graph – графы. Свойствами экземпляров класса являются: размерность (n\*n) (int), вектор вершин (vector<int>) и матрица смежности (vector<vector<T>>). Методы класса включают: конструкторы (без параметров, с параметрами, копирования), деструктор, селекторы, модификаторы, оператор сравнения, компонентные функции класса и дружественные функции класса.

Основные компонентные и дружественные функции класса:

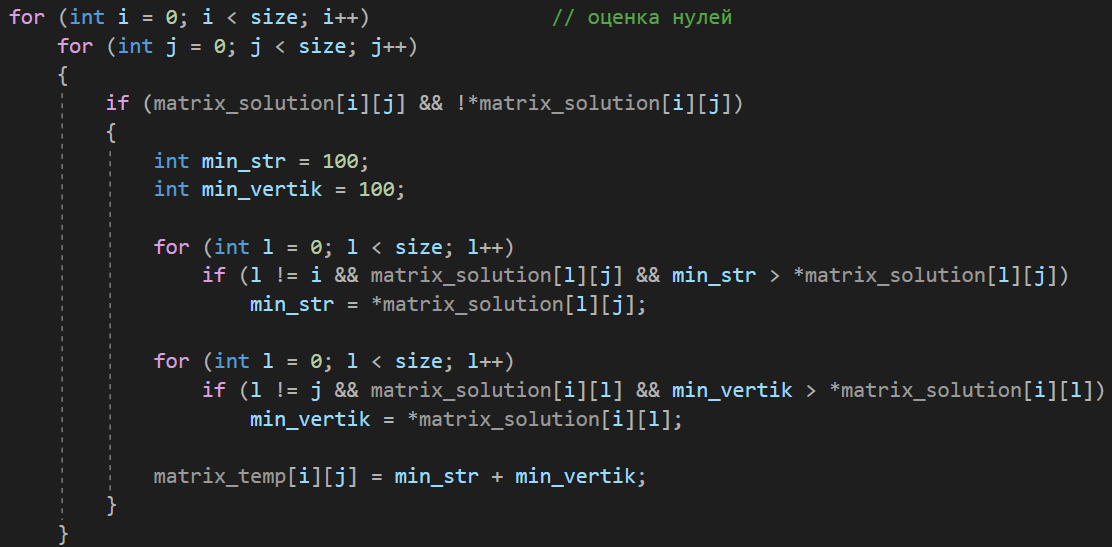
1. Функция void insert\_vertex(const T& new\_vertex) добавляет новую вершину в вектор вершин vertex\_List.push\_back(new\_vertex). Функция void insert\_edge(const T& vertex\_1, const T& vertex\_2, int value = 1) добавляет ребра (вес ребер) между заданными вершина в матрицу смежности. Для вставки ребра в матрицу: получение индексов заданных вершин int index\_vertex\_1 = get\_vertex\_index(vertex\_1); int index\_vertex\_2 = get\_vertex\_index(vertex\_2); так как граф ориентированный, то вес дорог зависит от направления движения путешественника => матрица смежности несимметрична => в двумерном векторе заполнение ячеек с указанными индексами adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] = value. Дополнительные проверки на: 1) существование заданных вершин; 2) повторное добавление ребра между вершинами.
2. Функция void DeleteVertex() удаляет вершину из вектора вершин vertex\_List.pop\_back() => преобразование матрицы смежности, то есть удаление строки и столбца данной вершины.
3. Функции void insert\_edge\_helicopter(), void insert\_edge\_scooter(), void insert\_edge\_car() преобразуют матрицу смежности графа, то есть на пересечении строки и столбца заданных городов уменьшают вес ребра в зависимости от функции ( -1 ч, -3 ч, -4 ч).
4. Решение задачи Коммивояжера – функция friend Graph<int> Kommivoyazher(Graph<int>& graph, int\*\*\* matrix\_solution, int\*\* matrix\_temp, int\* solution). Задача решается методом ветвей и границ. Метод содержит несколько этапов:
5. Нахождение минимального элемента в каждой строке(min\_di) и последующая редукция строк.



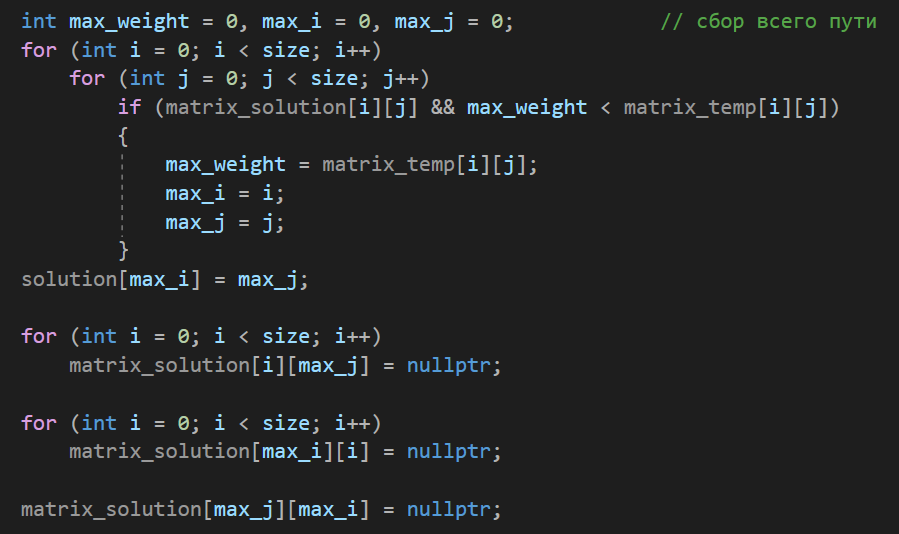
1. Нахождение минимального элемента в каждом столбце (min\_dj) и последующая редукция столбцов.



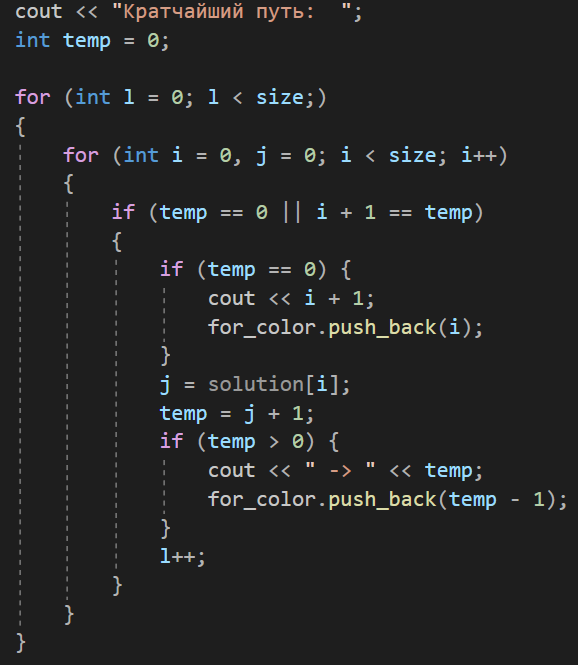
1. После редукций по строкам и столбцам, в каждой строке/столбце есть 0, следовательно повторять редукцию не имеет смысла, тк минимальным элементом будет 0. Далее производится оценка нулей, которая представляем собой сумму минимальных элементов в строке и столбце, содержащих ноль.



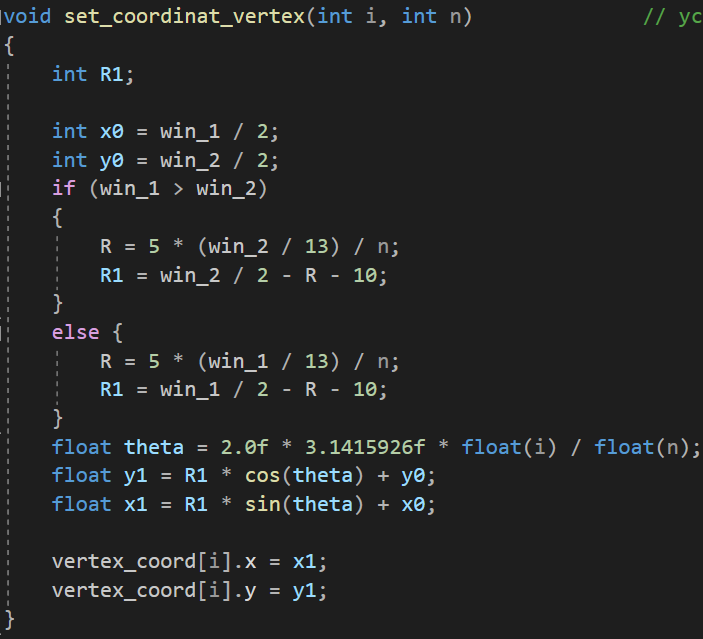
1. Поиск маршрута по оценкам нулей. В первую очередь берутся нули с максимальной оценкой, далее вычеркивание столбца и строки, содержащих данный ноль.



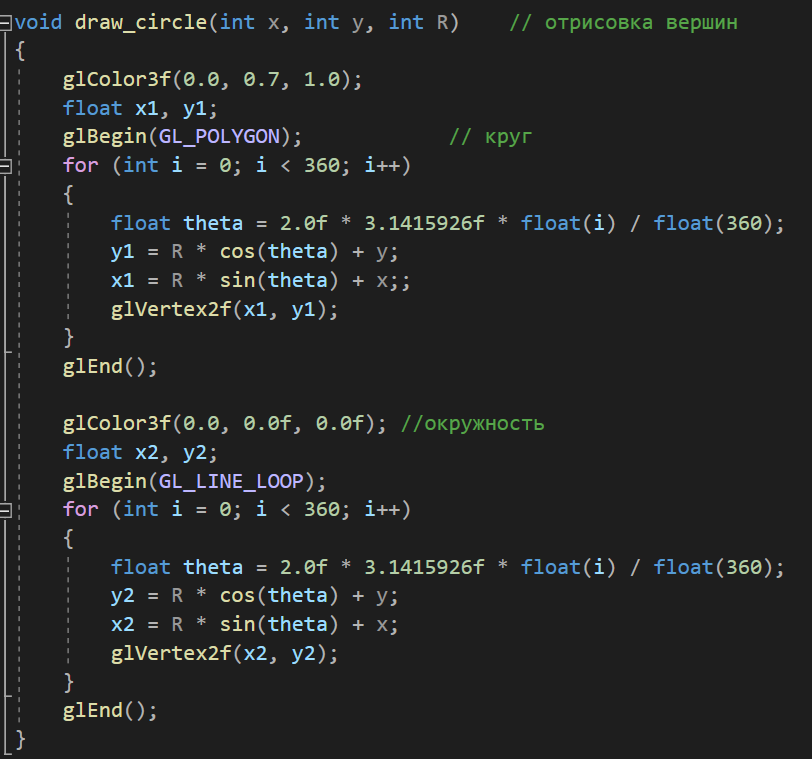
1. Вывод решения



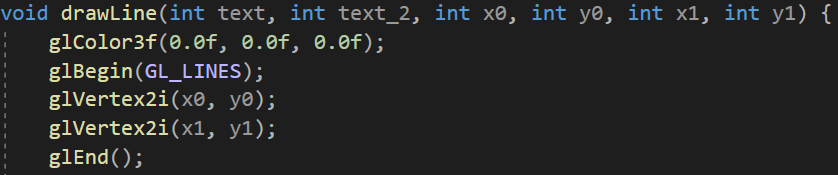
1. Функция void hours(int h\_2) для сравнения результатов работы классической задачи Коммивояжера и решения задачи с измененными условиями функциями void insert\_edge\_helicopter(), void insert\_edge\_scooter(), void insert\_edge\_car(). Данная функция возвращает разницу во времени пути.
2. Функция Graph<int> Graph\_new() для создания нового графа. Пользователь задает количество вершин и количество ребер графа, далее циклически происходит заполнение данных (вектора вершин и матрицы смежности)
3. Функция void Graph<T>::paint\_graph() рисует граф при помощи инструмента OpenGL(#include <glut.h>). В первую очередь вершинам графа устанавливаются координаты, вершины располагаются по окружности при помощи следующих формул:



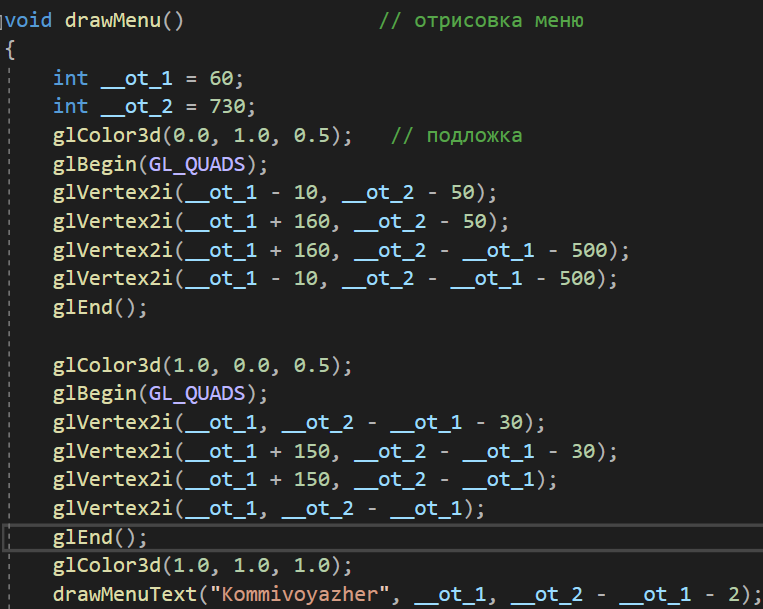
Координаты вершин хранятся в stract vertex\_coord. Далее по заданным координатам рисуются круги, которые являются вершинами графа:



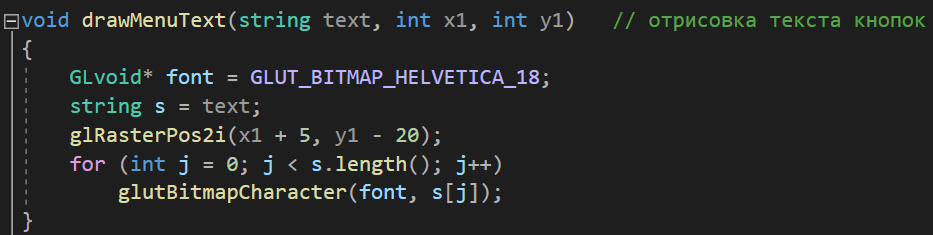
Последним этапом идет отрисовка ребер графа. Отрисовка ребра реализуется только в случае, если вес ребра не нулевой.



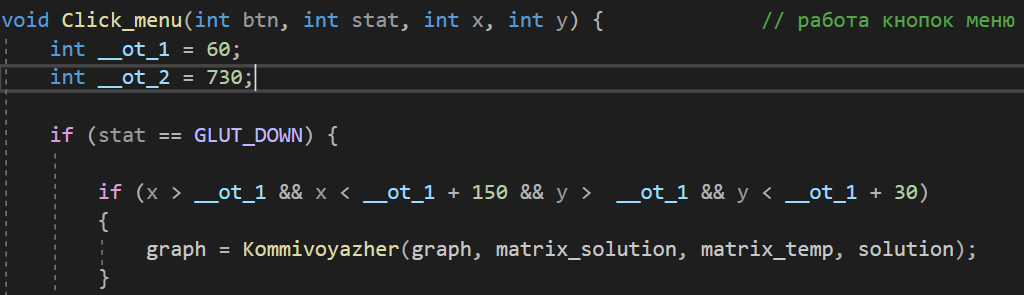
1. Функция для отрисовки меню программы void drawMenu(). Сначала отрисовка подложки в виде прямоугольника под кнопки меню, затем отрисовка самих кнопок прямоугольной формы.



Текст кнопок выводится функцией void drawMenuText(string text, int x1, int y1)



1. Функция для программирования каждой кнопки void Click\_menu(int btn, int stat, int x, int y). При клике на определенную кнопку происходит вывоз соответствующей функции.



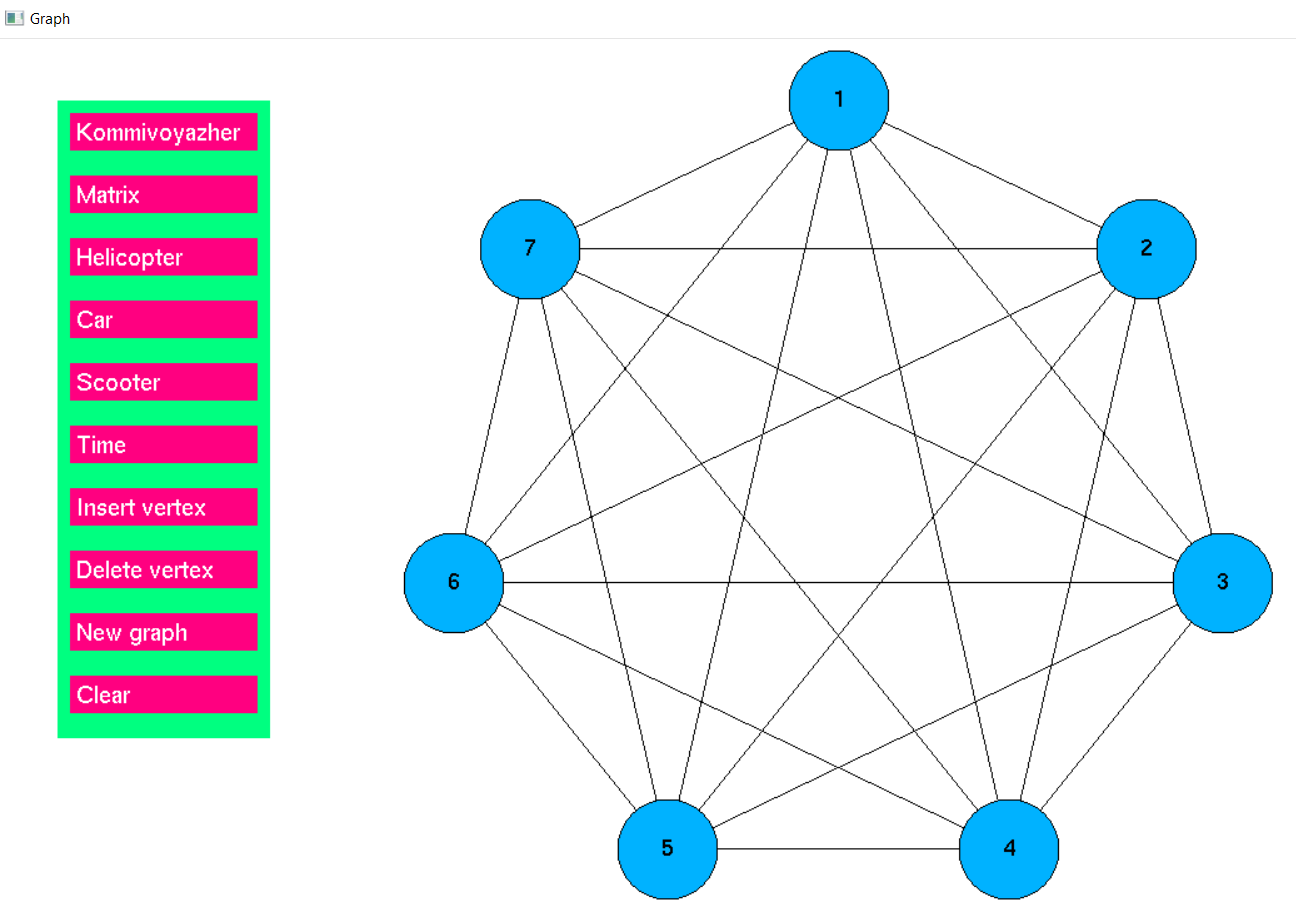
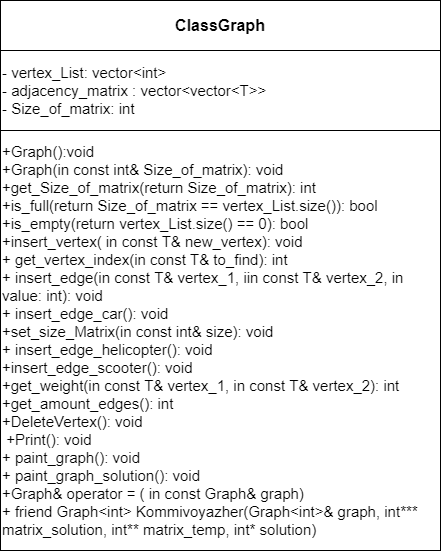
Дизайн программы выполнен в ярких оттенках на белом фоне, для привлечения внимания и заинтересованности пользователя. Рабочее окно программы представляет графическое изображение графа и меню программы с 10 функциями (рис. 3). При решении задачи Коммивояжера дороги, которые входят в самый быстрый по времени маршрут, подсвечиваются красным цветом. Ввод/вывод данных производится в консоли.

Рис.3 «Визуализация задачи Коммивояжера»

При выполнении творческого задания у меня были трудности с освоением инструмента OpenGL, но затратив достаточно времени я разобралась с базовыми функциями данного инструмента.

Особенностями и достоинствами данной программы являются:

1. Меню с десятью функциями для работы с графом (Коммивояжер, вывод матрицы смежности, удаление/ добавление вершины, создание нового графа, авто, вертолет, самокат, очистить решение, время).
2. Программа выводит начальных граф, который пользователь может изменять при помощи кнопок меню.
3. Решение задачи отображается на графе в виде путей красного цвета.
4. Имеются дополнительные функции, такие как: авто, вертолет, самокат; данные функции изменяют параметры классической задачи, в следствии этого можно уменьшить время пути/перестроить маршрут.

Диаграмма классов

Код программы

#include <iostream>

#include <vector>

#include <queue>

#include <glut.h>

#include <sstream>

using namespace std;

template<class T>

class Graph

{

private:

vector<int> vertex\_List;

vector<vector<T>> adjacency\_matrix;

int Size\_of\_matrix;

public:

Graph();

Graph(const int& Size\_of\_matrix);

int get\_Size\_of\_matrix() { return Size\_of\_matrix; }

bool is\_full() { return Size\_of\_matrix == vertex\_List.size(); }

bool is\_empty() { return vertex\_List.size() == 0; }

void insert\_vertex(const T& new\_vertex);

int get\_vertex\_index(const T& to\_find);

void insert\_edge(const T& vertex\_1, const T& vertex\_2, int value = 1);

void insert\_edge\_car();

void set\_size\_Matrix(const int& size) { this->Size\_of\_matrix = size; }

void insert\_edge\_helicopter();

void insert\_edge\_scooter();

int get\_weight(const T& vertex\_1, const T& vertex\_2);

int get\_amount\_edges();

void DeleteVertex();

void Print();

void paint\_graph();

void paint\_graph\_solution();

Graph& operator = (const Graph& graph) {

this->vertex\_List = graph.vertex\_List;

this->Size\_of\_matrix = graph.Size\_of\_matrix;

this->adjacency\_matrix = vector<vector<T>>(graph.Size\_of\_matrix, vector<T>(graph.Size\_of\_matrix));

for (int i = 0; i < graph.Size\_of\_matrix; ++i)

{

for (int j = 0; j < graph.Size\_of\_matrix; ++j)

{

this->adjacency\_matrix[i][j] = graph.adjacency\_matrix[i][j];

}

}

return \*this;

}

friend Graph<int> Kommivoyazher(Graph<int>& graph, int\*\*\* matrix\_solution, int\*\* matrix\_temp, int\* solution);

};

int R;

int house\_2;

int\*\* matrix\_temp;

int\* solution;

int\*\*\* matrix\_solution;

int win\_1;

int win\_2;

const int max\_size = 10;

struct coordinat\_vertex

{

int x, y;

};

coordinat\_vertex vertex\_coord[10];

Graph<int> graph;

Graph<int> graph\_2(7);

Graph<int> graph\_solution(7);

vector<int> for\_color;

void drawLine\_color(int x0, int y0, int x1, int y1);

void display();

template<class T>

Graph<T>::Graph() { // конструктор без параметров

this->Size\_of\_matrix = 0;

adjacency\_matrix = vector<vector<T>>(max\_size, vector<T>(max\_size));

for (int i = 0; i < this->Size\_of\_matrix; ++i)

{

for (int j = 0; j < this->Size\_of\_matrix; ++j)

{

if (i == j) {

adjacency\_matrix[i][j] = 10;

}

else {

adjacency\_matrix[i][j] = 0;

}

}

}

}

template<class T>

Graph<T>::Graph(const int& Size\_of\_matrix) // конструктор с параметрами

{

this->Size\_of\_matrix = Size\_of\_matrix;

adjacency\_matrix = vector<vector<T>>(Size\_of\_matrix, vector<T>(Size\_of\_matrix));

for (int i = 0; i < this->Size\_of\_matrix; ++i)

{

for (int j = 0; j < this->Size\_of\_matrix; ++j)

{

if (i == j) {

adjacency\_matrix[i][j] = 10;

}

else {

adjacency\_matrix[i][j] = 0;

}

}

}

}

template<class T>

int Graph<T>::get\_amount\_edges() // количество ребер графа

{

if (is\_empty()) return 0;

int amount = 0;

int size = vertex\_List.size();

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

{

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

{

if (adjacency\_matrix[i][j] != 0)

{

++amount;

}

}

}

return amount;

}

template<class T>

int Graph<T>::get\_weight(const T& vertex\_1, const T& vertex\_2) // вес ребра между вершинами

{

int index\_vertex\_1 = get\_vertex\_index(vertex\_1);

int index\_vertex\_2 = get\_vertex\_index(vertex\_2);

if (is\_empty() || index\_vertex\_1 == -1 || index\_vertex\_2 == -1) {// выход если вершин нет совсем или нет одной из вершин

cout << "Ошибка! Проверьте правильность ввода узлов графа!" << endl;

return 0;

}

return adjacency\_matrix[index\_vertex\_1][index\_vertex\_2];

}

template<class T>

int Graph<T>::get\_vertex\_index(const T& to\_find) // получение индекса заданной вершины

{

if (is\_empty()) return -1;

for (int i = 0; i < this->vertex\_List.size(); i++) {

if (this->vertex\_List[i] == to\_find) {

return i;

}

}

return -1;

}

template<class T>

void Graph<T>::insert\_vertex(const T& new\_vertex) // добавление вершины в вектор

{

if (is\_full())

{

cout << "Граф заполнен! Добавление новых вершин невозможно." << endl;

return;

}

vertex\_List.push\_back(new\_vertex);

}

template<class T>

void Graph<T>::DeleteVertex() { // удаление вершины

int h = this->get\_Size\_of\_matrix();

this->vertex\_List.pop\_back();

h--;

this->set\_size\_Matrix(h);

for (int i = 0; i < this->Size\_of\_matrix; ++i)

{

for (int j = 0; j < this->Size\_of\_matrix; ++j)

{

this->adjacency\_matrix[i][j] = adjacency\_matrix[i][j];

}

}

}

template<class T>

void Graph<T>::insert\_edge(const T& vertex\_1, const T& vertex\_2, int value) // вставка вершины

{

if ((get\_vertex\_index(vertex\_1) == -1 || get\_vertex\_index(vertex\_2) == -1))

{

cout << "Ошибка! Не удалется добавить ребро. " << endl;

return;

}

int index\_vertex\_1 = get\_vertex\_index(vertex\_1);

int index\_vertex\_2 = get\_vertex\_index(vertex\_2);

if (adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] != 0)

{

cout << "Ребро между вершинами уже существует." << endl;

return;

}

adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] = value;

}

template<class T>

void Graph<T>::insert\_edge\_car() // доп. возможность сократить путь между городами

{

int from, to;

cout << "Коммивояжер передвигается на АВТО из города "; cin >> from; cout << " в город "; cin >> to;

if ((get\_vertex\_index(from) == -1 || get\_vertex\_index(to) == -1))

{

cout << "Ошибка! Не существует одного города или двух городов. " << endl;

return;

}

int index\_vertex\_1 = get\_vertex\_index(from);

int index\_vertex\_2 = get\_vertex\_index(to);

adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] = adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] - 3;

this->Print();

}

template<class T>

void Graph<T>::insert\_edge\_helicopter() // доп. возможность сократить путь между городами

{

int from, to;

cout << "Коммивояжер передвигается на ВЕРТОЛЕТЕ из города "; cin >> from; cout << " в город "; cin >> to;

if ((get\_vertex\_index(from) == -1 || get\_vertex\_index(to) == -1))

{

cout << "Ошибка! Не существует одного города или двух городов. " << endl;

return;

}

int index\_vertex\_1 = get\_vertex\_index(from);

int index\_vertex\_2 = get\_vertex\_index(to);

adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] = adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] - 4;

this->Print();

}

template<class T>

void Graph<T>::insert\_edge\_scooter() // доп. возможность сократить путь между городами

{

int from, to;

cout << "Коммивояжер передвигается на ЭЛЕКТРОСАМОКАТЕ из города "; cin >> from; cout << " в город "; cin >> to;

if ((get\_vertex\_index(from) == -1 || get\_vertex\_index(to) == -1))

{

cout << "Ошибка! Не существует одного города или двух городов. " << endl;

return;

}

int index\_vertex\_1 = get\_vertex\_index(from);

int index\_vertex\_2 = get\_vertex\_index(to);

adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] = adjacency\_matrix[index\_vertex\_1][index\_vertex\_2] - 1;

this->Print();

}

template<class T>

void Graph<T>::Print() // вывод матрицы смежности

{

if (!this->is\_empty()) {

cout << "Матрица смежности графа: " << endl;

cout << " ";

for (int i = 0; i < vertex\_List.size(); i++)

{

cout << vertex\_List[i] << " ";

}

cout << endl;

for (int i = 0; i < this->vertex\_List.size(); i++) {

cout << this->vertex\_List[i] << " ";

for (int j = 0; j < this->vertex\_List.size(); j++) {

cout << " " << this->adjacency\_matrix[i][j] << " ";

}

cout << endl;

}

}

else {

cout << "Граф не существует! " << endl;

}

}

Graph<int> Kommivoyazher(Graph<int>& graph, int\*\*\* matrix\_solution, int\*\* matrix\_temp, int\* solution) {

int size = graph.get\_Size\_of\_matrix();

matrix\_temp = new int\* [size];

solution = new int[size];

matrix\_solution = new int\*\* [size];

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

{

matrix\_temp[i] = new int[size];

}

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

{

matrix\_solution[i] = new int\* [size];

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

{

if (graph.adjacency\_matrix[i][j] == 0) {

matrix\_solution[i][j] = nullptr;

}

matrix\_solution[i][j] = new int(graph.adjacency\_matrix[i][j]);

}

}

for (int l = 0; l < size; l++) // редукция строк

{

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

{

int min\_di = 100;

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

if (matrix\_solution[i][j] && min\_di > \*matrix\_solution[i][j])

min\_di = \*matrix\_solution[i][j];

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

if (matrix\_solution[i][j])

\*matrix\_solution[i][j] -= min\_di;

}

for (int j = 0; j < size; j++) // редукция столбцов

{

int min\_dj = 100;

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

if (matrix\_solution[i][j] && min\_dj > \*matrix\_solution[i][j])

min\_dj = \*matrix\_solution[i][j];

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

if (matrix\_solution[i][j])

\*matrix\_solution[i][j] -= min\_dj;

}

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

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

matrix\_temp[i][j] = 0;

for (int i = 0; i < size; i++) // оценка нулей

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

{

if (matrix\_solution[i][j] && !\*matrix\_solution[i][j])

{

int min\_str = 100;

int min\_vertik = 100;

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

if (l != i && matrix\_solution[l][j] && min\_str > \*matrix\_solution[l][j])

min\_str = \*matrix\_solution[l][j];

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

if (l != j && matrix\_solution[i][l] && min\_vertik > \*matrix\_solution[i][l])

min\_vertik = \*matrix\_solution[i][l];

matrix\_temp[i][j] = min\_str + min\_vertik;

}

}

int max\_weight = 0, max\_i = 0, max\_j = 0; // сбор всего пути

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

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

if (matrix\_solution[i][j] && max\_weight < matrix\_temp[i][j])

{

max\_weight = matrix\_temp[i][j];

max\_i = i;

max\_j = j;

}

solution[max\_i] = max\_j;

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

matrix\_solution[i][max\_j] = nullptr;

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

matrix\_solution[max\_i][i] = nullptr;

matrix\_solution[max\_j][max\_i] = nullptr;

}

//cout << "Отрезки пути ";

int s = 0;

for (int i = 0, j = 0; i < size; i++)

{

j = solution[i];

//cout << i + 1 << " -> " << j + 1 << '\t';

s += graph.adjacency\_matrix[i][j];

}

cout << endl;

cout << "Кратчайший путь: ";

int temp = 0;

for (int l = 0; l < size;)

{

for (int i = 0, j = 0; i < size; i++)

{

if (temp == 0 || i + 1 == temp)

{

if (temp == 0) {

cout << i + 1;

for\_color.push\_back(i);

}

j = solution[i];

temp = j + 1;

if (temp > 0) {

cout << " -> " << temp;

for\_color.push\_back(temp - 1);

}

l++;

}

}

}

house\_2 = s;

cout << endl << "Время в пути: " << s<<" ч."<<endl;

for (int i = 0; i < graph.Size\_of\_matrix; ++i)

{

for (int j = 0; j < graph.Size\_of\_matrix; ++j)

{

if (i != j) {

graph\_solution.adjacency\_matrix[i][j] = graph.adjacency\_matrix[i][j];

}

else {

graph\_solution.adjacency\_matrix[i][j] = 10;

}

}

}

graph\_solution.vertex\_List = graph.vertex\_List; //отметка дороги в матрице смежности для визуализации на графе(ребра красного цвета)

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

{

for (int j = i + 1; j < for\_color.size() - 1; j += size - j + i)

{

graph\_solution.adjacency\_matrix[for\_color[i]][for\_color[j]] = -1;

}

}

graph\_solution.adjacency\_matrix[for\_color[for\_color.size()-1]][for\_color[0]] = -1;

cout << endl;

for\_color.clear();

return graph\_solution;

}

template<class T>

Graph<T> create\_graph(Graph<T>& graph) // создание графа

{

graph.set\_size\_Matrix(7);

for (int i = 1; i < 8; i++) {

graph.insert\_vertex(i);

}

graph.insert\_edge(1, 2, 6);

graph.insert\_edge(1, 3, 9);

graph.insert\_edge(1, 4, 1);

graph.insert\_edge(1, 5, 7);

graph.insert\_edge(1, 6, 2);

graph.insert\_edge(1, 7, 5);

graph.insert\_edge(2, 1, 9);

graph.insert\_edge(2, 3, 1);

graph.insert\_edge(2, 4, 6);

graph.insert\_edge(2, 5, 4);

graph.insert\_edge(2, 6, 3);

graph.insert\_edge(2, 7, 7);

graph.insert\_edge(3, 1, 2);

graph.insert\_edge(3, 2, 5);

graph.insert\_edge(3, 4, 9);

graph.insert\_edge(3, 5, 9);

graph.insert\_edge(3, 6, 8);

graph.insert\_edge(3, 7, 9);

graph.insert\_edge(4, 1, 3);

graph.insert\_edge(4, 2, 9);

graph.insert\_edge(4, 3, 7);

graph.insert\_edge(4, 5, 2);

graph.insert\_edge(4, 6, 8);

graph.insert\_edge(4, 7, 6);

graph.insert\_edge(5, 1, 8);

graph.insert\_edge(5, 2, 8);

graph.insert\_edge(5, 3, 4);

graph.insert\_edge(5, 4, 9);

graph.insert\_edge(5, 6, 3);

graph.insert\_edge(5, 7, 9);

graph.insert\_edge(6, 1, 5);

graph.insert\_edge(6, 2, 9);

graph.insert\_edge(6, 3, 1);

graph.insert\_edge(6, 4, 9);

graph.insert\_edge(6, 5, 5);

graph.insert\_edge(6, 7, 7);

graph.insert\_edge(7, 1, 2);

graph.insert\_edge(7, 2, 7);

graph.insert\_edge(7, 3, 9);

graph.insert\_edge(7, 4, 9);

graph.insert\_edge(7, 5, 8);

graph.insert\_edge(7, 6, 9);

return graph;

}

void hours(int h\_2) { // подсчет разницы времени

int house\_1 = 21;

cout << "Время в пути сократилось на " << house\_1 - h\_2 << " ч." << endl;

}

void set\_coordinat\_vertex(int i, int n) // установка координат для вершин графа для визуализации

{

int R1;

int x0 = win\_1 / 2;

int y0 = win\_2 / 2;

if (win\_1 > win\_2)

{

R = 5 \* (win\_2 / 13) / n;

R1 = win\_2 / 2 - R - 10;

}

else {

R = 5 \* (win\_1 / 13) / n;

R1 = win\_1 / 2 - R - 10;

}

float theta = 2.0f \* 3.1415926f \* float(i) / float(n);

float y1 = R1 \* cos(theta) + y0;

float x1 = R1 \* sin(theta) + x0;

vertex\_coord[i].x = x1;

vertex\_coord[i].y = y1;

}

void draw\_circle(int x, int y, int R) // отрисовка вершин

{

glColor3f(0.0, 0.7, 1.0);

float x1, y1;

glBegin(GL\_POLYGON); // круг

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0, 0.0f, 0.0f); //окружность

float x2, y2;

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 360; i++)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void draw\_text\_for\_edge(int text\_1, int text\_2, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(text\_1);

glRasterPos2i(x1 - 10, y1 - 10);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void draw\_text\_for\_vertex(int text, int x1, int y1) // отрисовка номеров вершин

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(text);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void draw\_vertex(int n) // отрисовка вершин

{

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

draw\_circle(vertex\_coord[i].x, vertex\_coord[i].y, R);

draw\_text\_for\_vertex(i + 1, vertex\_coord[i].x, vertex\_coord[i].y);

}

}

void drawLine(int text, int text\_2, int x0, int y0, int x1, int y1) { // отрисовка ребер

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

//draw\_text\_for\_edge(text, text\_2, (x0 + x1) / 2 + 15, (y0 + y1) / 2 + 15);

}

void drawLine\_color(int x0, int y0, int x1, int y1) { // отрисовка ребер при решении Коммивояжера ( дорога - красный)

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

}

template<class T>

void Graph<T>::paint\_graph() // отрисовка графа в целом

{

int n = vertex\_List.size();

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

{

set\_coordinat\_vertex(i, n);

}

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

{

for (int j = i + 1; j < n; j++)

{

int a = adjacency\_matrix[i][j];

int weight\_2 = adjacency\_matrix[j][i];

if (a != 0|| weight\_2!=0)

{

drawLine(a, weight\_2, vertex\_coord[i].x, vertex\_coord[i].y, vertex\_coord[j].x, vertex\_coord[j].y);

}

if (a == -1 || weight\_2 == -1) {

drawLine\_color(vertex\_coord[i].x, vertex\_coord[i].y, vertex\_coord[j].x, vertex\_coord[j].y);

}

}

}

draw\_vertex(n);

}

template<class T>

void Graph<T>::paint\_graph\_solution()

{

int n = vertex\_List.size();

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

{

set\_coordinat\_vertex(i, n);

}

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

{

for (int j = i + 1; j < n; j++)

{

int a = adjacency\_matrix[i][j];

int weight\_2;

if (a != 0)

{

drawLine(a, weight\_2, vertex\_coord[i].x, vertex\_coord[i].y, vertex\_coord[j].x, vertex\_coord[j].y);

}

if (a == -1) {

drawLine\_color(vertex\_coord[i].x, vertex\_coord[i].y, vertex\_coord[j].x, vertex\_coord[j].y);

}

}

}

draw\_vertex(n);

}

Graph<int> Graph\_new() // создание нового графа

{

int edge\_count, first\_vertex, second\_vertex, weight\_edge, vertex\_count;

cout << "Введите количество вершин графа: ";

cin >> vertex\_count;

Graph<int> graph(vertex\_count);

cout << endl;

cout << "Введите количество ребер графа: ";

cin >> edge\_count; cout << endl;

for (int i = 1; i <= vertex\_count; ++i) {

int\* vertPtr = &i;

graph.insert\_vertex(\*vertPtr);

}

for (int i = 0; i < edge\_count; ++i) {

cout << "ИСХОДНАЯ вершина: "; cin >> first\_vertex; cout << endl;

int\* first\_vertex\_ptr = &first\_vertex;

cout << "КОНЕЧНАЯ вершина: "; cin >> second\_vertex; cout << endl;

int\* second\_vertex\_ptr = &second\_vertex;

cout << "Вес ребра: "; cin >> weight\_edge; cout << endl;

graph.insert\_edge(\*first\_vertex\_ptr, \*second\_vertex\_ptr, weight\_edge);

}

cout << endl;

return graph;

}

void drawMenuText(string text, int x1, int y1) // отрисовка текста кнопок

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = text;

glRasterPos2i(x1 + 5, y1 - 20);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawMenu() // отрисовка меню

{

int \_\_ot\_1 = 60;

int \_\_ot\_2 = 730;

glColor3d(0.0, 1.0, 0.5); // подложка

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1 - 10, \_\_ot\_2 - 50);

glVertex2i(\_\_ot\_1 + 160, \_\_ot\_2 - 50);

glVertex2i(\_\_ot\_1 + 160, \_\_ot\_2 - \_\_ot\_1 - 500);

glVertex2i(\_\_ot\_1 - 10, \_\_ot\_2 - \_\_ot\_1 - 500);

glEnd();

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 30);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 30);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Kommivoyazher", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 2);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 80);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 80);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 50);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 50);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Matrix", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 52);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 130);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 130);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 100);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 100);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Helicopter", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 102);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 180);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 180);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 150);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 150);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Car", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 152);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 230);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 230);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 200);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 200);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Scooter", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 202);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 280);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 280);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 250);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 250);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Time", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 252);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 330);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 330);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 300);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 300);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Insert vertex", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 302);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 380);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 380);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 350);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 350);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Delete vertex", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 352);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 430);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 430);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 400);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 400);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("New graph", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 402);

glColor3d(1.0, 0.0, 0.5);

glBegin(GL\_QUADS);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 480);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 480);

glVertex2i(\_\_ot\_1 + 150, \_\_ot\_2 - \_\_ot\_1 - 450);

glVertex2i(\_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 450);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Clear", \_\_ot\_1, \_\_ot\_2 - \_\_ot\_1 - 452);

}

void Click\_menu(int btn, int stat, int x, int y) { // работа кнопок меню

int \_\_ot\_1 = 60;

int \_\_ot\_2 = 730;

if (stat == GLUT\_DOWN) {

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 && y < \_\_ot\_1 + 30)

{

graph = Kommivoyazher(graph, matrix\_solution, matrix\_temp, solution);

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 50 && y < \_\_ot\_1 + 80)

{

graph.Print();

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 100 && y < \_\_ot\_1 + 130)

{

graph.insert\_edge\_helicopter();

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 150 && y < \_\_ot\_1 + 180)

{

graph.insert\_edge\_car();

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 200 && y < \_\_ot\_1 + 230)

{

graph.insert\_edge\_scooter();

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 250 && y < \_\_ot\_1 + 280)

{

hours(house\_2);

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 300 && y < \_\_ot\_1 + 330)

{

int first\_vertex;

int second\_vertex;

int weight\_edge;

int size\_m = graph.get\_Size\_of\_matrix();

cout << "ИСХОДНАЯ вершина: "; cin >> first\_vertex; cout << endl;

int\* firsr\_vertex\_ptr = &first\_vertex;

cout << "КОНЕЧНАЯ вершина: "; cin >> second\_vertex; cout << endl;

int\* second\_vertex\_ptr = &second\_vertex;

if (first\_vertex > size\_m || second\_vertex > size\_m) {

size\_m++;

int\* vertPtr = &size\_m;

graph.set\_size\_Matrix(size\_m);

graph.insert\_vertex(\*vertPtr);

}

cout << "Вес ребра: "; cin >> weight\_edge; cout << endl;

graph.insert\_edge(\*firsr\_vertex\_ptr, \*second\_vertex\_ptr, weight\_edge);

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 350 && y < \_\_ot\_1 + 380)

{

int count\_vertex = graph.get\_Size\_of\_matrix();

int first\_vertex;

cout << "Введите вершину: "; cin >> first\_vertex; cout << endl;

int\* first\_vertex\_ptr = &first\_vertex;

if (first\_vertex == count\_vertex)

{

count\_vertex--;

graph.DeleteVertex();

}

else cout << "Невозможно удалить вершину \n";

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 400 && y < \_\_ot\_1 + 430)

{

graph = Graph\_new();

}

if (x > \_\_ot\_1 && x < \_\_ot\_1 + 150 && y > \_\_ot\_1 + 450 && y < \_\_ot\_1 + 480)

{

graph = graph\_2;

}

}

glutPostRedisplay();

}

void display() // картинка окна, окно

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, win\_1, 0, win\_2);

glViewport(0, 0, win\_1, win\_2);

glClearColor(1.0, 1.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.paint\_graph();

drawMenu();

glutSwapBuffers();

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "RU");

create\_graph(graph\_2);

create\_graph(graph);

graph.Print();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1350, 730);

glutCreateWindow("Graph");

win\_1 = glutGet(GLUT\_WINDOW\_WIDTH);

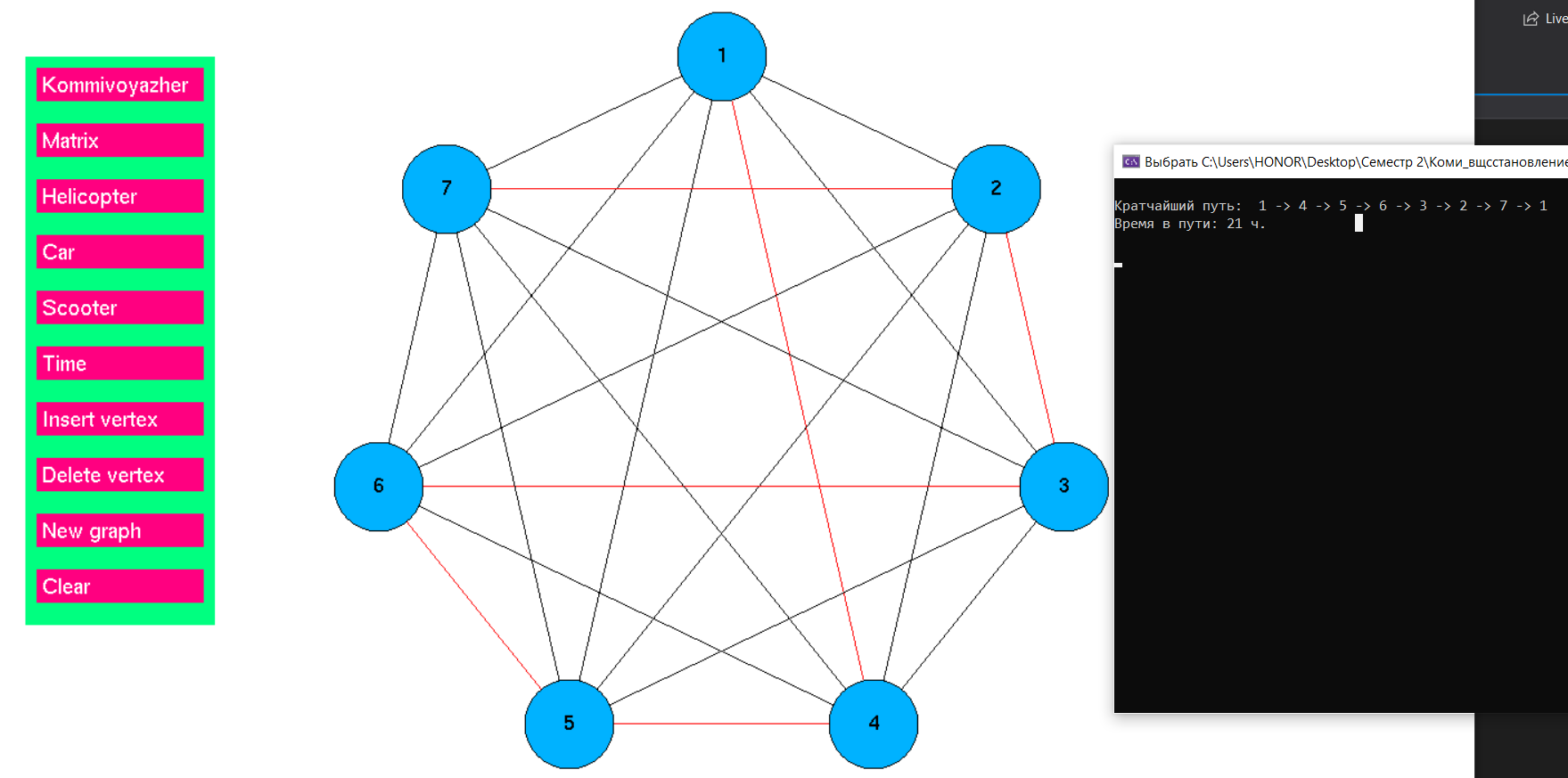
win\_2 = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutMouseFunc(Click\_menu);

glutMainLoop();

Скриншоты результатов



Решение задачи Коммивояжера для данного графа проделано «руками» с помощью таблицы Excel. Результаты совпадают.

