Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Творческая работа   
«Решение задачи Коммивояжера»**

Выполнил:

Студент группы РИС-23-1б

Шароглазов Е. А

Мокрушин Н. Д

Проверила:

Доцент кафедры ИТАС

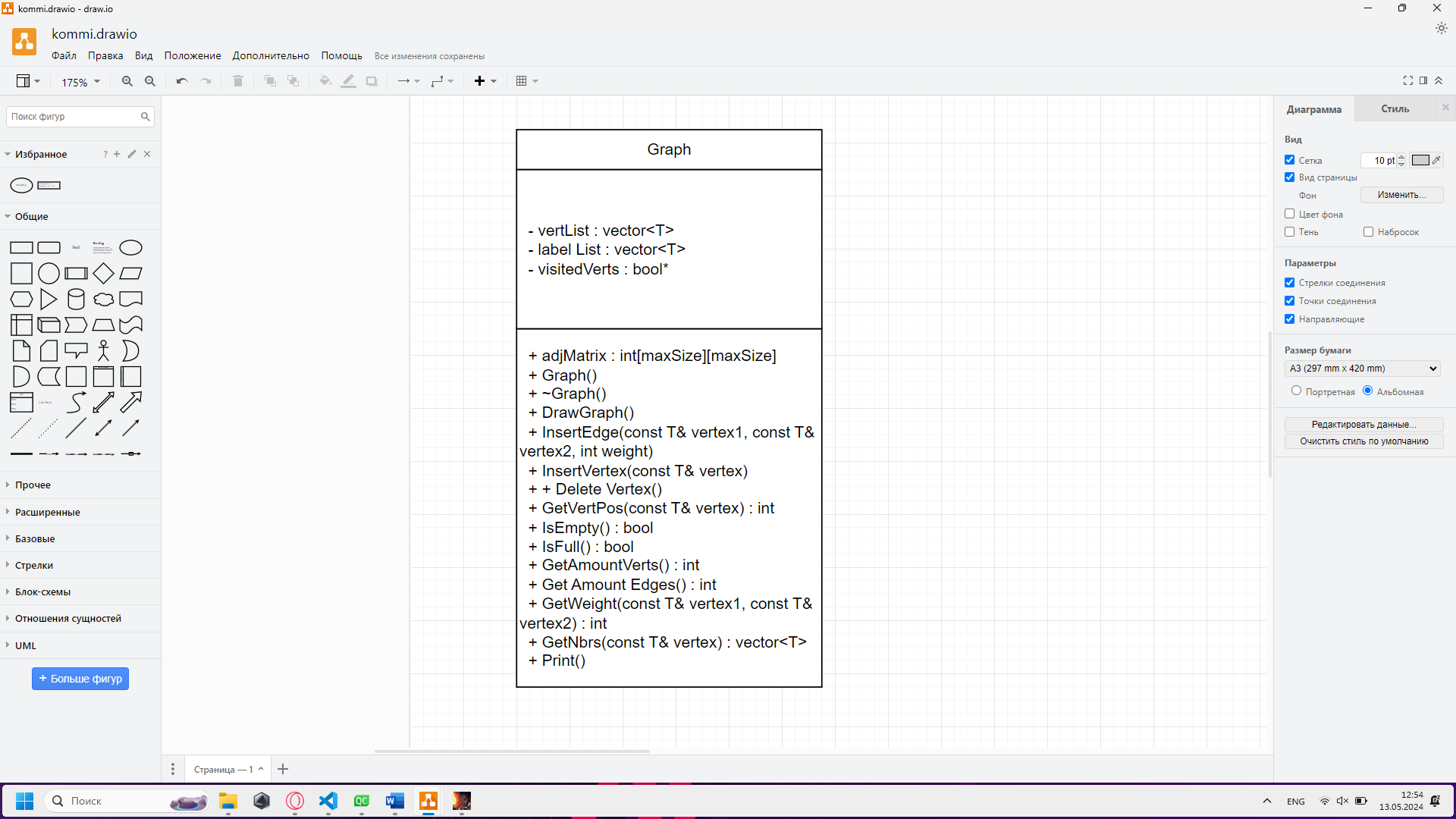
О.А. Полякова

Пермь 2024

**Постановка задачи Коммивояжера:**

Построить произвольный граф. Рассчитать самый короткий путь между вершинами с возвратом в исходную, не проходя через одну и ту же вершину дважды.

**UML – Диаграмма**



**Код:**

**Graph.h**

**// Подключение необходимых заголовочных файлов**

**#pragma once**

**#include "GL/glut.h" // Заголовочный файл GLUT для работы с графикой**

**#include <stdio.h>**

**#include <iostream>**

**#include <vector>**

**#include <sstream>**

**using namespace std;**

**// Глобальные переменные**

**int n; // Количество вершин в графе**

**int\*\* help; // Вспомогательный массив для алгоритма**

**int\* result; // Массив для хранения результата алгоритма**

**int\*\*\* mat; // Массив для хранения матрицы смежности графа**

**int WinW; // Ширина окна**

**int WinH; // Высота окна**

**const int maxSize = 20; // Максимальный размер графа**

**int amountVerts; // Количество вершин**

**// Объявление шаблонного класса Graph для работы с графом**

**template<class T>**

**class Graph**

**{**

**vector<T> vertList;**

**vector<T> labelList;**

**bool\* visitedVerts = new bool[vertList.size()];**

**public:**

**int adjMatrix[maxSize][maxSize] = { 0 };**

**Graph();**

**~Graph();**

**void DrawGraph();**

**void InsertEdge(const T& vertex1, const T& vertex2, int weight);**

**void InsertVertex(const T& vertex);**

**void DeleteVertex();**

**int GetVertPos(const T& vertex);**

**bool IsEmpty();**

**bool IsFull();**

**int GetAmountVerts();**

**int GetAmountEdges();**

**int GetWeight(const T& vertex1, const T& vertex2);**

**vector<T> GetNbrs(const T& vertex);**

**void Print();**

**};**

**int R;//радиус**

**// координаты**

**struct vertCoord**

**{**

**int x, y;**

**};**

**vertCoord vertC[20];**

**Graph<int> graph;//объект класса**

**// Функция для выполнения алгоритма коммивояжера**

**void answer(int\*\*\* mat, int n, int\*\* help, int\* path)**

**{**

**for (int l = 0; l < n; l++)**

**{**

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

**{**

**int min = 1000000;**

**for (int j = 0; j < n; j++)**

**if (mat[i][j] && min > \*mat[i][j])**

**min = \*mat[i][j];**

**for (int j = 0; j < n; j++)**

**if (mat[i][j])**

**\*mat[i][j] -= min;**

**}**

**for (int j = 0; j < n; j++)**

**{**

**int min = 1000000;**

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

**if (mat[i][j] && min > \*mat[i][j])**

**min = \*mat[i][j];**

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

**if (mat[i][j])**

**\*mat[i][j] -= min;**

**}**

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

**for (int j = 0; j < n; j++)**

**help[i][j] = 0;**

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

**for (int j = 0; j < n; j++)**

**{**

**if (mat[i][j] && !\*mat[i][j])**

**{**

**int hmin = 1000000;**

**int vmin = 1000000;**

**for (int l = 0; l < n; l++)**

**if (l != i && mat[l][j] && hmin > \*mat[l][j])**

**hmin = \*mat[l][j];**

**for (int l = 0; l < n; l++)**

**if (l != j && mat[i][l] && vmin > \*mat[i][l])**

**vmin = \*mat[i][l];**

**help[i][j] = hmin + vmin;**

**}**

**}**

**int mcost = 0, mi = 0, mj = 0;**

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

**for (int j = 0; j < n; j++)**

**if (mat[i][j] && mcost < help[i][j])**

**{**

**mcost = help[i][j];**

**mi = i;**

**mj = j;**

**}**

**path[mi] = mj;**

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

**mat[i][mj] = nullptr;**

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

**mat[mi][i] = nullptr;**

**mat[mj][mi] = nullptr;**

**}**

**}**

**// Подготовка данных для алгоритма коммивояжера**

**void preparation(int\*\*\*& mat, int& n, int\*\*& help, int\*& result)**

**{**

**n = amountVerts;**

**help = new int\* [n];**

**result = new int[n];**

**mat = new int\*\* [n];**

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

**{**

**help[i] = new int[n];**

**}**

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

**{**

**mat[i] = new int\* [n];**

**for (int j = 0; j < n; j++)**

**{**

**if (graph.adjMatrix[i][j] == 0) {**

**mat[i][j] = nullptr;**

**continue;**

**}**

**mat[i][j] = new int(graph.adjMatrix[i][j]);**

**}**

**}**

**}**

**// Функция для выполнения шага алгоритма коммивояжера**

**void tsalesman(int\*\*\* mat, int n, int\*\* help, int\* result)**

**{**

**preparation(mat, n, help, result);**

**int s = 0;**

**answer(mat, n, help, result);**

**cout << "\nОтрезки путей: ";**

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

**{**

**j = result[i];**

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

**s += graph.adjMatrix[i][j];**

**}**

**cout << endl;**

**cout << "\nКратчайший путь: ";**

**int temp = 0;**

**for (int l = 0; l < n;)**

**{**

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

**{**

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

**{**

**if (temp == 0) cout << i + 1;**

**j = result[i];**

**temp = j + 1;**

**if (temp > 0) cout << " -> " << temp;**

**l++;**

**}**

**}**

**}**

**cout << "\nМинимальное расстояние: " << s;**

**cout << endl;**

**}**

**//соседи от узла**

**template<class T>**

**vector<T> Graph<T>::GetNbrs(const T& vertex) {**

**vector<T> nbrsList; // создание списка соседних вершин**

**int vertPos = this->GetVertPos(vertex); // вычисление позиции vertex в матрице смежности**

**if (vertPos != (-1)) {**

**for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i) {**

**if (this->adjMatrix[vertPos][i] != 0 &&**

**this->adjMatrix[i][vertPos] != 0) // вычисление соседних вершин**

**nbrsList.push\_back(this->vertList[i]);**

**}**

**}**

**return nbrsList;**

**}**

**// вставка узла**

**template<class T>**

**void Graph<T>::InsertVertex(const T& vertex) {**

**if (!this->IsFull()) {**

**this->vertList.push\_back(vertex);**

**}**

**else {**

**cout << "Граф запполнен" << endl;**

**return;**

**}**

**}**

**//удаление узла**

**template<class T>**

**void Graph<T>::DeleteVertex() {**

**this->vertList.pop\_back();**

**}**

**// количество ребер**

**template<class T>**

**int Graph<T>::GetAmountEdges() {**

**int amount = 0; // обнуляем счётчик рёбер**

**if (!this->IsEmpty()) { // если граф не пустой**

**for (int i = 0, vertListSize = this->vertList.size();**

**i < vertListSize; ++i) {**

**for (int j = 0; j < vertListSize; ++j) {**

**if (this->adjMatrix[i][j] ==**

**this->adjMatrix[j][i] &&**

**this->adjMatrix[i][j] != 0) // находим рёбра**

**amount += 1; // увеличиваем количество рёбер**

**}**

**}**

**return (amount / 2); // возвращаем счётчик**

**}**

**else**

**return 0; // если же граф пустой то возвращаем 0 рёбер**

**}**

**//вес между узлами**

**template<class T>**

**int Graph<T>::GetWeight(const T& vertex1, const T& vertex2) {**

**if (!this->IsEmpty()) {**

**int vertPos1 = GetVertPos(vertex1);**

**int vertPos2 = GetVertPos(vertex2);**

**return adjMatrix[vertPos1][vertPos2];**

**}**

**return 0;**

**}**

**//Метод для получения количества вершин в графе**

**template<class T>**

**int Graph<T>::GetAmountVerts() {**

**return this->vertList.size();**

**}**

**// если пуст**

**template<class T>**

**bool Graph<T>::IsEmpty() {**

**if (this->vertList.size() != 0)**

**return false;**

**else**

**return true;**

**}**

**//если полон**

**template<class T>**

**bool Graph<T>::IsFull() {**

**return (vertList.size() == maxSize);**

**}**

**// позиция вершины**

**template <class T>**

**int Graph<T>::GetVertPos(const T& vertex) {**

**for (int i = 0; i < this->vertList.size(); ++i) {**

**if (this->vertList[i] == vertex)**

**return i;**

**}**

**return -1;**

**}**

**template<class T>**

**Graph<T>::Graph() {**

**for (int i = 0; i < maxSize; ++i)**

**{**

**for (int j = 0; j < maxSize; ++j)**

**{**

**this->adjMatrix[i][j] = 0;**

**}**

**}**

**}**

**template<class T>**

**Graph<T>::~Graph() {**

**}**

**//метод создания графа**

**Graph<int> makeGraph()**

**{**

**Graph<int> graph; // создание графа**

**int amountEdges, sourceVertex, targetVertex, edgeWeight; // создание необходимых для ввода графа переменных**

**cout << "Количество вершин >> "; cin >> amountVerts; cout << endl; // ввод количества вершин графа в переменную amountVerts**

**cout << "Количество рёбер >> : "; cin >> amountEdges; cout << endl; // ввод количества рёбер графа в переменную amountEdges**

**for (int i = 1; i <= amountVerts; ++i) {**

**int\* vertPtr = &i; // запоминаем адрес вершины**

**graph.InsertVertex(\*vertPtr); //передаём ссылку на вершину в функцию InsertVertex; происходит вставка вершины в вектор вершин**

**}**

**for (int i = 0; i < amountEdges; ++i) {**

**cout << "Начальная вершина >> "; cin >> sourceVertex; cout << endl; // ввод исходной вершины**

**int\* sourceVertPtr = &sourceVertex; // запоминаем адрес исходной вершины**

**cout << "Конечная вершина >> "; cin >> targetVertex; cout << endl; // ввод вершины, до которой будет идти ребро от исходной вершины**

**int\* targetVertPtr = &targetVertex; // запоминаем адрес конечной вершины (до которой будет идти ребро от исходной вершины)**

**cout << "Вес >> "; cin >> edgeWeight; cout << endl; // ввод числового значения веса ребра в переменную edgeWeight**

**graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight); // вставка ребра весом edgeWeight между исходной и конечной вершинами**

**}**

**cout << endl;**

**return graph;**

**}**

**//вставка ребра**

**template<class T>**

**void Graph<T>::InsertEdge(const T& vertex1, const T& vertex2, int weight) {**

**if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1)) {**

**int vertPos1 = GetVertPos(vertex1);**

**int vertPos2 = GetVertPos(vertex2);**

**if (this->adjMatrix[vertPos1][vertPos2] != 0**

**&& this->adjMatrix[vertPos2][vertPos1] != 0) {**

**cout << "Данное ребро уже есть" << endl;**

**return;**

**}**

**else {**

**this->adjMatrix[vertPos1][vertPos2] = weight;**

**this->adjMatrix[vertPos2][vertPos1] = weight;**

**}**

**}**

**else {**

**cout << "Этих вершин нет в графе" << endl;**

**return;**

**}**

**}**

**//вывод матрицы смежности**

**template<class T>**

**void Graph<T>::Print() {**

**if (!this->IsEmpty()) {**

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

**for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i) {**

**cout << this->vertList[i] << " ";**

**for (int j = 0; j < vertListSize; ++j) {**

**cout << " " << this->adjMatrix[i][j] << " ";**

**}**

**cout << endl;**

**}**

**}**

**}**

**//задать координаты**

**void setCoord(int i, int n)**

**{**

**int R\_;**

**int x0 = WinW / 2;**

**int y0 = WinH / 2;**

**if (WinW > WinH)**

**{**

**R = 5 \* (WinH / 13) / n;**

**R\_ = WinH / 2 - R - 10;**

**}**

**else {**

**R = 5 \* (WinW / 13) / n;**

**R\_ = WinW / 2 - R - 10;**

**}**

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

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

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

**vertC[i].x = x1;**

**vertC[i].y = y1;**

**}**

**void drawCircle(int x, int y, int R) //рисуем круг в заданных координатах**

**{**

**glColor3f(0.7529, 0.7529, 0.7529);**

**float x1, y1;**

**glBegin(GL\_POLYGON);**

**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(0.0f, 0.0f, 0.0f);**

**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();**

**}**

**//риусем текст**

**void drawText(int nom, int x1, int y1)**

**{**

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

**string s = to\_string(nom);**

**glRasterPos2i(x1 - 5, y1 - 5);**

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

**glutBitmapCharacter(font, s[j]);**

**}**

**//рисуем вершину**

**void drawVertex(int n)**

**{**

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

**drawCircle(vertC[i].x, vertC[i].y, R);**

**drawText(i + 1, vertC[i].x, vertC[i].y);**

**}**

**}**

**//рисуем линию**

**void drawLine(int text, int x0, int y0, int x1, int y1)**

**{**

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

**glBegin(GL\_LINES);**

**glVertex2i(x0, y0);**

**glVertex2i(x1, y1);**

**glEnd();**

**drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);**

**}**

**//отрисовка графа**

**template<class T>**

**void Graph<T>::DrawGraph()**

**{**

**int n = vertList.size();**

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

**{**

**setCoord(i, n);**

**}**

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

**{**

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

**{**

**int a = adjMatrix[i][j];**

**if (a != 0)**

**{**

**drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);**

**}**

**}**

**}**

**drawVertex(n);**

**}**

**//функция для glutreshape**

**void reshape(int w, int h)**

**{**

**WinW = w;**

**WinH = h;**

**glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);**

**glutPostRedisplay();**

**}**

**//отрисовка текста меню**

**void drawMenuText(string text, int x1, int y1)**

**{**

**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]);**

**}**

**//отрисовка меню**

**void drawMenu()**

**{**

**int shift = 60;**

**int height = 730;**

**glColor3d(0.0, 0.0, 0.0);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 10, height - 50);**

**glVertex2i(shift + 145, height - 50);**

**glVertex2i(shift + 145, height - shift - 200);**

**glVertex2i(shift - 10, height - shift - 200);**

**glEnd();**

**glColor3d(0.0, 0.55, 0.55);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 5, height - shift - 35);**

**glVertex2i(shift + 140, height - shift - 35);**

**glVertex2i(shift + 140, height - shift + 5);**

**glVertex2i(shift - 5, height - shift + 5);**

**glEnd();**

**glColor3d(1, 1, 1);**

**drawMenuText("Insert element", shift, height - shift - 2);**

**glColor3d(0.0, 0.55, 0.55);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 5, height - shift - 75);**

**glVertex2i(shift + 140, height - shift - 75);**

**glVertex2i(shift + 140, height - shift - 40);**

**glVertex2i(shift - 5, height - shift - 40);**

**glEnd();**

**glColor3d(1, 1, 1);**

**drawMenuText("Delete element", shift, height - shift - 42);**

**glColor3d(0.0, 0.55, 0.55);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 5, height - shift - 115);**

**glVertex2i(shift + 140, height - shift - 115);**

**glVertex2i(shift + 140, height - shift - 80);**

**glVertex2i(shift - 5, height - shift - 80);**

**glEnd();**

**glColor3d(1, 1, 1);**

**drawMenuText("Print matrix", shift, height - shift - 82);**

**glColor3d(0.0, 0.55, 0.55);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 5, height - shift - 155);**

**glVertex2i(shift + 140, height - shift - 155);**

**glVertex2i(shift + 140, height - shift - 120);**

**glVertex2i(shift - 5, height - shift - 120);**

**glEnd();**

**glColor3d(1, 1, 1);**

**drawMenuText("Answer", shift, height - shift - 122);**

**glColor3d(0.0, 0.55, 0.55);**

**glBegin(GL\_QUADS);**

**glVertex2i(shift - 5, height - shift - 195);**

**glVertex2i(shift + 140, height - shift - 195);**

**glVertex2i(shift + 140, height - shift - 160);**

**glVertex2i(shift - 5, height - shift - 160);**

**glEnd();**

**glColor3d(1, 1, 1);**

**drawMenuText("New matrix", shift, height - shift - 162);**

**}**

**//клик по меню по координатам**

**void mouseClick(int btn, int stat, int x, int y) {**

**int shift = 60;**

**int height = 730;**

**if (stat == GLUT\_DOWN) {**

**if (x > shift && x < shift + 135 && y > shift && y < shift + 30)**

**{**

**int sourceVertex;**

**int targetVertex;**

**int edgeWeight;**

**cout << "Начальная вершина >> "; cin >> sourceVertex; cout << endl;**

**int\* sourceVertPtr = &sourceVertex;**

**cout << "Конечная вершина >> "; cin >> targetVertex; cout << endl;**

**int\* targetVertPtr = &targetVertex;**

**if (sourceVertex > amountVerts || targetVertex > amountVerts) {**

**amountVerts++;**

**int\* vertPtr = &amountVerts;**

**graph.InsertVertex(\*vertPtr);**

**}**

**cout << "Вес >> "; cin >> edgeWeight; cout << endl;**

**graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight); //**

**}**

**if (x > shift && x < shift + 135 && y > shift + 40 && y < shift + 70)**

**{**

**int sourceVertex;**

**int targetVertex;**

**int edgeWeight;**

**cout << "Удалить вершину >> "; cin >> sourceVertex; cout << endl;**

**int\* sourceVertPtr = &sourceVertex;**

**if (sourceVertex == amountVerts)**

**{**

**amountVerts--;**

**graph.DeleteVertex();**

**}**

**else cout << "Такой вершины нет!\n";**

**}**

**if (x > shift && x < shift + 135 && y > shift + 80 && y < shift + 100)**

**{**

**graph.Print();**

**}**

**if (x > shift && x < shift + 135 && y > shift + 120 && y < shift + 140)**

**{**

**tsalesman(mat, n, help, result);**

**}**

**if (x > shift && x < shift + 135 && y > shift + 160 && y < shift + 180)**

**{**

**graph = makeGraph();**

**}**

**}**

**glutPostRedisplay();**

**}**

**//функция для glutdisplay**

**void display()**

**{**

**glShadeModel(GL\_SMOOTH);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**gluOrtho2D(0, WinW, 0, WinH); //ставим начало координат в левый нижний угол**

**glViewport(0, 0, WinW, WinH);**

**glClearColor(0.421875, 0.484375, 0.59375, 1.0);**

**glClear(GL\_COLOR\_BUFFER\_BIT);**

**graph.DrawGraph();**

**drawMenu();**

**glutSwapBuffers();**

**}**

**Graph.cpp**

#include "Graph.h"

#include "GL/glut.h"

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

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

{

setlocale(LC\_ALL, "rus");

glutInit(&argc, argv);

graph = makeGraph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1350, 730);

glutCreateWindow("Graph");

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouseClick);

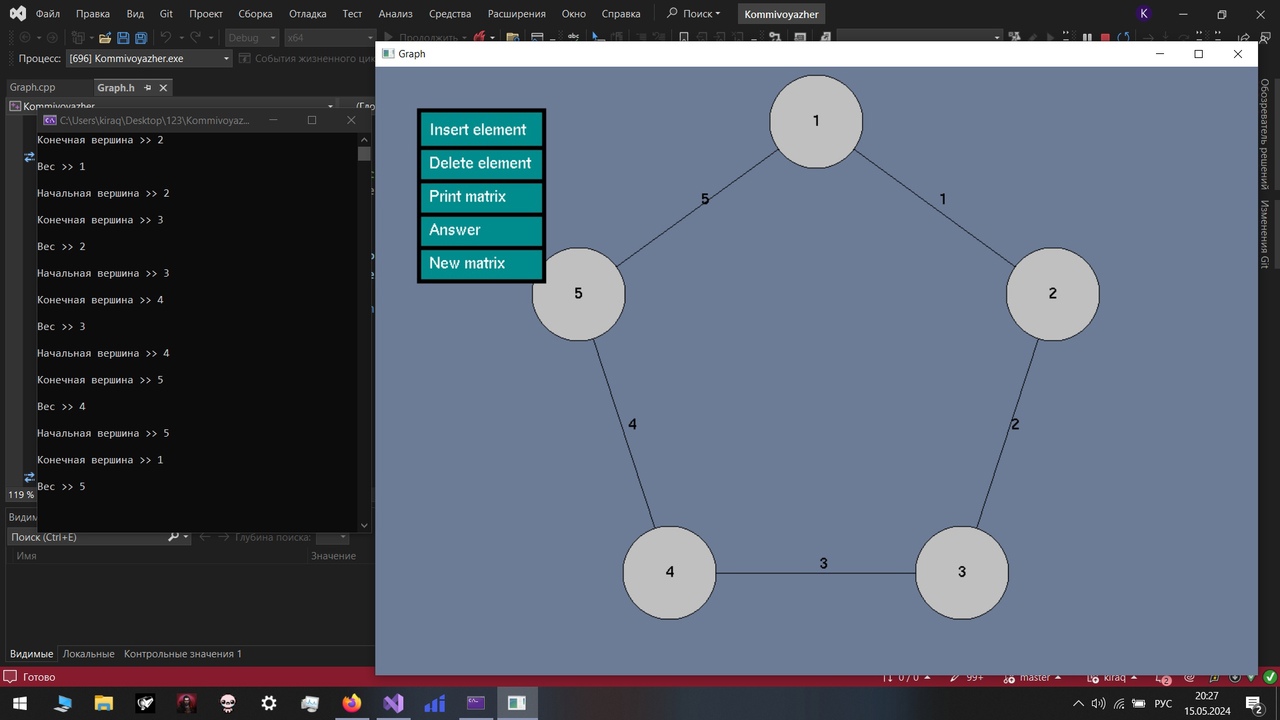
glutMainLoop();

return 0;

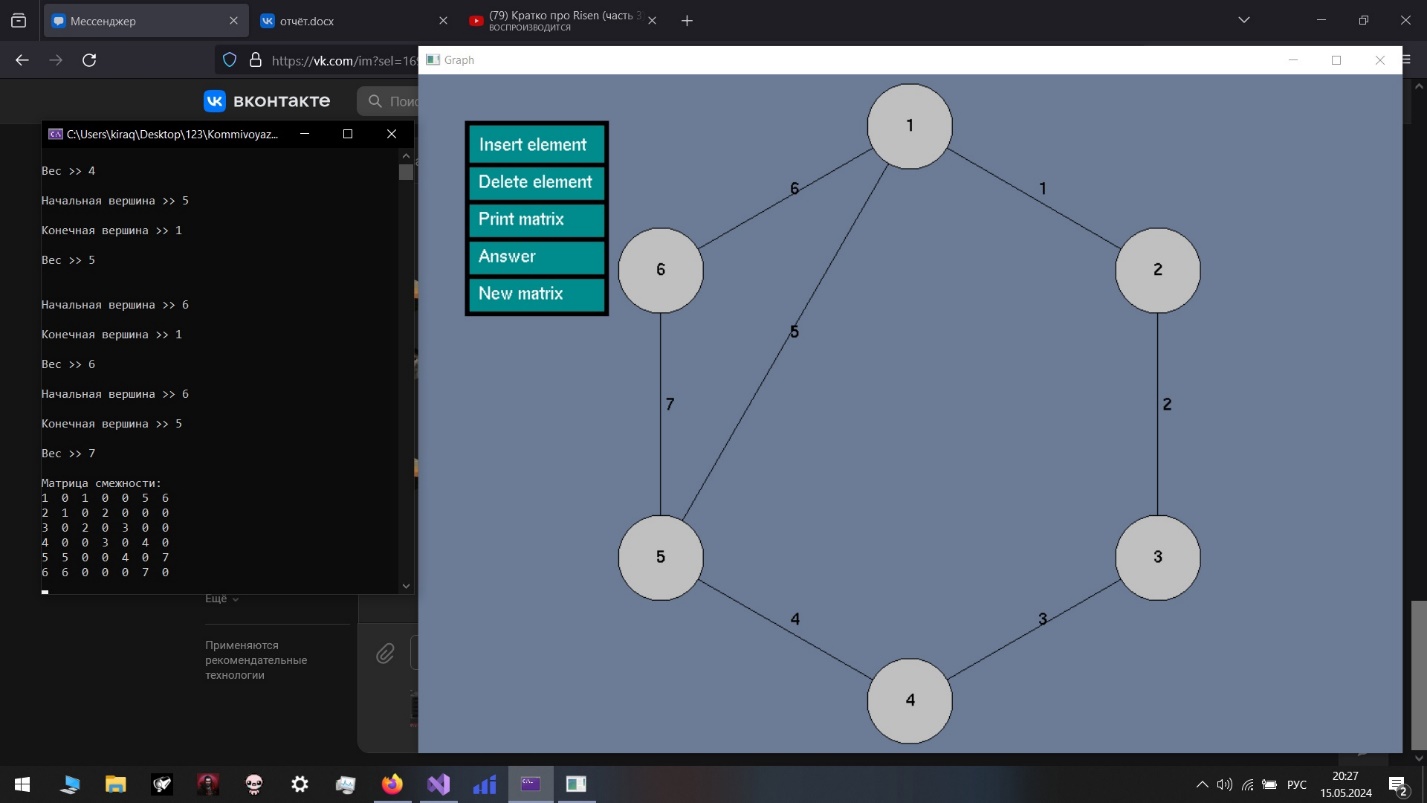
}

**Примеры работы программы**

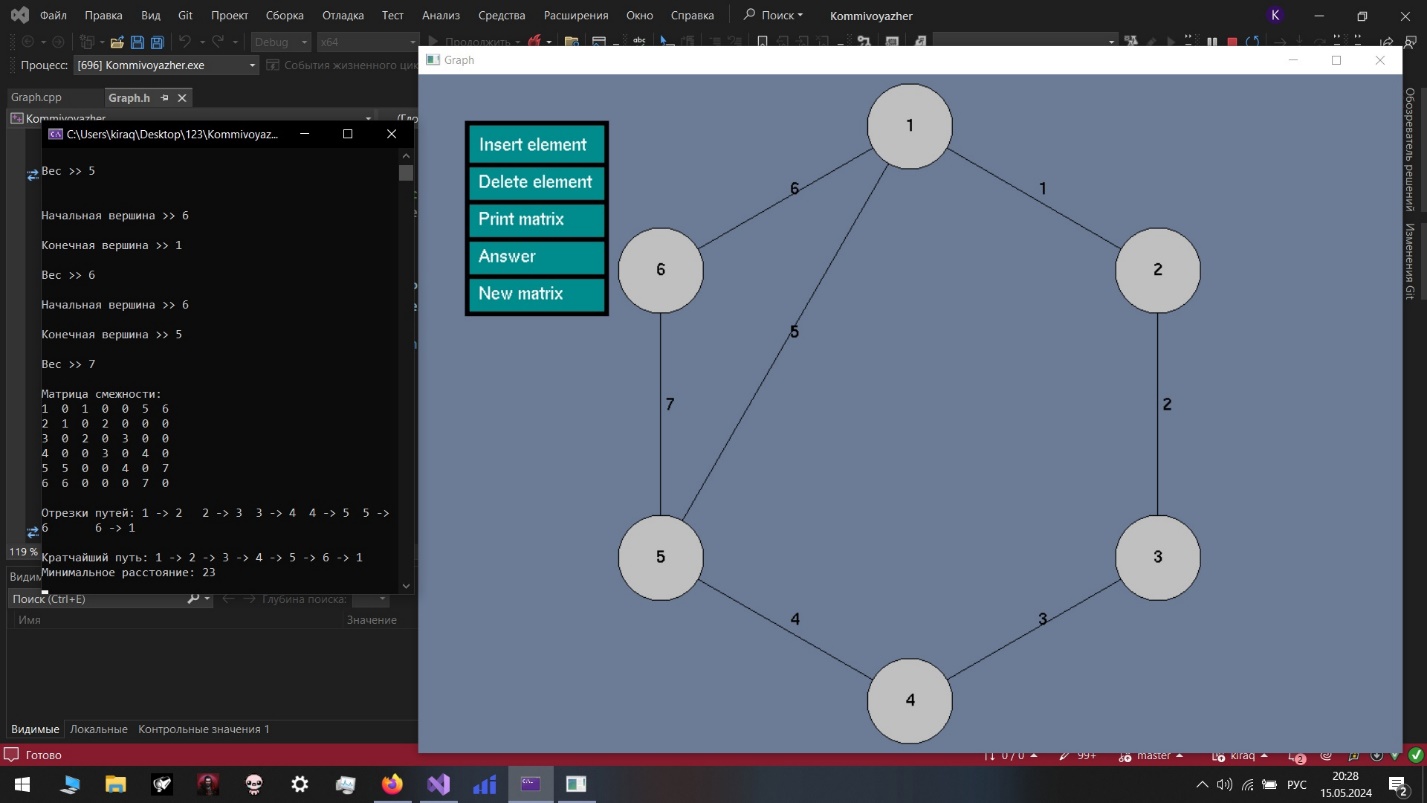
1. Создание графа



1. Вывод матрицы смежности



1. Вывод ответа на задачу Коммивояжера



1. Удаление вершины графа

