**Пермский национальный исследовательский  
политехнический университет**

**О Т Ч Ё Т**

**по творческой работе №2**

Задача коммивояжера

Дисциплина: информатика

Вариант: 21

Выполнил работу

студент группы РИС-20-1б

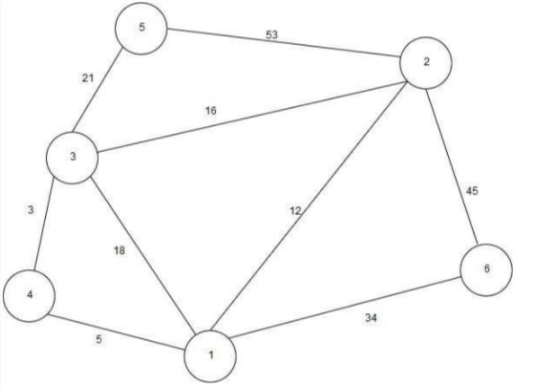
Копейкин Дмитрий Евгеньевич

Проверила

Доцент кафедры ИТАС Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

Функция решения задачи Коммивояжера

void answer\_of\_kom(int\*\*\* mat, int n, int\*\* help, int\* result)

{

pre(mat, n, help, result);

int s = 0;

post(mat, n, help, result);

cout << "\n������� �����: ";

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 << "\n���������� ����: ";

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 << "\n����������: " << s << " �������� ������";

cout << endl;

}

Функция построения графа

Graph<int> makeGraph()

{

Graph<int> graph;

int amountEdges, sourceVertex, targetVertex, edgeWeight;

cout << "������� ���������� ������ �����: "; cin >> amountVerts; cout << endl;

cout << "������� ���������� ����� �����: "; cin >> amountEdges; cout << endl;

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

int\* vertPtr = &i;

graph.AddVertex(\*vertPtr);

}

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

cout << "��������� �������: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "�������� �������: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "��� ����: "; cin >> edgeWeight; cout << endl;

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

}

cout << endl;

return graph;

}

Функция вывода матрицы смежности

template<class T>

void Graph<T>::Print() {

if (!this->IsEmpty()) {

cout << "���� ������� ���������: " << endl;

for (int i = 0, verticListSize = this->verticList.size(); i < verticListSize; ++i) {

cout << this->verticList[i] << " ";

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

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

}

cout << endl;

}

}

}

Функция рисование «Городов»

void drawCircle(int x, int y, int R)

{

glColor3f(0.4, 0.6, 0.0);

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();

}

Функция вывода графа на экран

template<class T>

void Graph<T>::DrawGraph()

{

int n = verticList.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);

}

Функции визуализации меню и действия кнопок

void drawMenu()

{

int shift = 50;

int height = 730;

glColor3d(0.6, 0.2, 0.0);

glBegin(GL\_QUADS);

glVertex2i(shift - 20, height - 40);

glVertex2i(shift + 145, height - 40);

glColor3d(0.6, 0.2, 1.0);

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

glVertex2i(shift - 20, height - shift - 200);

glEnd();

glColor3d(0.0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 30);

glVertex2i(shift + 135, height - shift - 30);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift);

glVertex2i(shift, height - shift);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Print matrix", shift, height - shift - 2);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 70);

glVertex2i(shift + 135, height - shift - 70);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 40);

glVertex2i(shift, height - shift - 40);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Del element", shift, height - shift - 42);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 110);

glVertex2i(shift + 135, height - shift - 110);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 80);

glVertex2i(shift, height - shift - 80);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Add element", shift, height - shift - 82);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 150);

glVertex2i(shift + 135, height - shift - 150);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 120);

glVertex2i(shift, height - shift - 120);

glEnd();

glColor3d(0.8, 1.0, 0.9);

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

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 190);

glVertex2i(shift + 135, height - shift - 190);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 160);

glVertex2i(shift, height - shift - 160);

glEnd();

glColor3d(0.8, 1.0, 0.9);

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

}

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

int shift = 50;

int height = 630;

if (stat == GLUT\_DOWN) {

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

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "��������� �������: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "�������� �������: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

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

amountVerts++;

int\* vertPtr = &amountVerts;

graph.AddVertex(\*vertPtr);

}

cout << "��� ������: "; 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;

sourceVertex = amountVerts;

int\* sourceVertPtr = &sourceVertex;

amountVerts--;

graph.DelVertex();

}

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

{

graph.Print();

}

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

{

answer\_of\_kom(mat, n, help, result);

}

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

{

graph = makeGraph();

}

}

glutPostRedisplay();

}

**Код**

**Visual.h**

#pragma once

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

#include <glut.h>

using namespace std;

int n, WinW, WinH, amountVerts;

int\*\* help;

int\* result;

int\*\*\* mat;

const int maxSize = 20;

template<class T>

class Graph

{

vector<T> labelList;

vector<T> verticList;

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

public:

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

Graph();

~Graph();

void DrawGraph();

void AddVertex(const T& vertex);

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

void DelVertex();

bool IsFull();

bool IsEmpty();

int GetAmountVerts();

int GetAmountEdges();

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

int GetVertPos(const T& vertex);

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

void Print();

};

int R;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

void pre(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]);

}

}

}

void post(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;

}

}

void answer\_of\_kom(int\*\*\* mat, int n, int\*\* help, int\* result)

{

pre(mat, n, help, result);

int s = 0;

post(mat, n, help, result);

cout << "\n������� �����: ";

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 << "\n���������� ����: ";

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 << "\n����������: " << s << " �������� ������";

cout << endl;

}

template<class T>

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

std::vector<T> nbrsList;

int vertPos = this->GetVertPos(vertex);

if (vertPos != (-1)) {

for (int i = 0, verticListSize = this->verticList.size(); i < verticListSize; ++i) {

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

this->adjMatrix[i][vertPos] != 0)

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

}

}

return nbrsList;

}

template<class T>

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

if (!this->IsFull()) {

this->verticList.push\_back(vertex);

}

else {

cout << "���� ����������. ���������� �������� ����� �������." << endl;

return;

}

}

template<class T>

void Graph<T>::DelVertex() {

this->verticList.pop\_back();

}

template<class T>

int Graph<T>::GetAmountEdges() {

int amount = 0;

if (!this->IsEmpty()) {

for (int i = 0, verticListSize = this->verticList.size();

i < verticListSize; ++i) {

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

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

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

this->adjMatrix[i][j] != 0)

amount += 1;

}

}

return (amount / 2);

}

else

return 0;

}

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;

}

template<class T>

int Graph<T>::GetAmountVerts() {

return this->verticList.size();

}

template<class T>

bool Graph<T>::IsEmpty() {

if (this->verticList.size() != 0)

return false;

else

return true;

}

template<class T>

bool Graph<T>::IsFull() {

return (verticList.size() == maxSize);

}

template <class T>

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

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

if (this->verticList[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() {

}

Graph<int> makeGraph()

{

Graph<int> graph;

int amountEdges, sourceVertex, targetVertex, edgeWeight;

cout << "������� ���������� ������ �����: "; cin >> amountVerts; cout << endl;

cout << "������� ���������� ����� �����: "; cin >> amountEdges; cout << endl;

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

int\* vertPtr = &i;

graph.AddVertex(\*vertPtr);

}

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

cout << "��������� �������: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "�������� �������: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "��� ����: "; cin >> edgeWeight; cout << endl;

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

}

cout << endl;

return graph;

}

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 << "????? ????? ????????? ??? ????" << endl;

return;

}

else {

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

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

}

}

else {

cout << "������ �������� ����" << endl;

return;

}

}

template<class T>

void Graph<T>::Print() {

if (!this->IsEmpty()) {

cout << "���� ������� ���������: " << endl;

for (int i = 0, verticListSize = this->verticList.size(); i < verticListSize; ++i) {

cout << this->verticList[i] << " ";

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

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

}

cout << endl;

}

}

}

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)

{

glColor3f(0.4, 0.6, 0.0);

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();

}

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]);

}

void drawVertex(int n)

{

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

glColor3f(0.2456f, 0.6574f, 0.5437f);

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

glColor3f(0.2456f, 0.1574f, 0.5437f);

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

}

}

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

{

glColor3f(0.7543f, 0.8456f, 0.4321f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glColor3f(0.3673f, 0.1533f, 0.7853f);

glVertex2i(x1, y1);

glEnd();

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

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

}

template<class T>

void Graph<T>::DrawGraph()

{

int n = verticList.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);

}

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();

}

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]);

}

void drawMenu()

{

int shift = 50;

int height = 730;

glColor3d(0.6, 0.2, 0.0);

glBegin(GL\_QUADS);

glVertex2i(shift - 20, height - 40);

glVertex2i(shift + 145, height - 40);

glColor3d(0.6, 0.2, 1.0);

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

glVertex2i(shift - 20, height - shift - 200);

glEnd();

glColor3d(0.0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 30);

glVertex2i(shift + 135, height - shift - 30);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift);

glVertex2i(shift, height - shift);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Print matrix", shift, height - shift - 2);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 70);

glVertex2i(shift + 135, height - shift - 70);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 40);

glVertex2i(shift, height - shift - 40);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Del element", shift, height - shift - 42);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 110);

glVertex2i(shift + 135, height - shift - 110);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 80);

glVertex2i(shift, height - shift - 80);

glEnd();

glColor3d(0.8, 1.0, 0.9);

drawMenuText("Add element", shift, height - shift - 82);

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 150);

glVertex2i(shift + 135, height - shift - 150);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 120);

glVertex2i(shift, height - shift - 120);

glEnd();

glColor3d(0.8, 1.0, 0.9);

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

glColor3d(0, 0.3, 0.7);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 190);

glVertex2i(shift + 135, height - shift - 190);

glColor3d(0.5, 0.3, 0.0);

glVertex2i(shift + 135, height - shift - 160);

glVertex2i(shift, height - shift - 160);

glEnd();

glColor3d(0.8, 1.0, 0.9);

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

}

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

int shift = 50;

int height = 630;

if (stat == GLUT\_DOWN) {

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

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "��������� �������: "; cin >> sourceVertex; cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "�������� �������: "; cin >> targetVertex; cout << endl;

int\* targetVertPtr = &targetVertex;

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

amountVerts++;

int\* vertPtr = &amountVerts;

graph.AddVertex(\*vertPtr);

}

cout << "��� ������: "; 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;

sourceVertex = amountVerts;

int\* sourceVertPtr = &sourceVertex;

amountVerts--;

graph.DelVertex();

}

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

{

graph.Print();

}

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

{

answer\_of\_kom(mat, n, help, result);

}

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

{

graph = makeGraph();

}

}

glutPostRedisplay();

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(0.4745, 0.5529, 0.4352, 1.0); glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3d(0.0862, 0.0529, 0.4352);

glBegin(GL\_QUADS);

glVertex2i(0, 730);

glVertex2i(1350, 730);

glColor3d(0.9862, 0.5529, 0.6352);

glVertex2i(1350, 0);

glColor3d(0.9862, 0.5529, 0.0352);

glVertex2i(0, 0);

glEnd();

graph.DrawGraph();

drawMenu();

glutSwapBuffers();

}

void visual()

{

}

**Main.cpp**

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

#include "Visual.h"

#include <glut.h>

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;

}

**Скриншот**

