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

ПНИПУ

**Лабораторная работа  
Бинарные деревья**

Выполнил:   
студент группы РИС-23-1б   
Молодых Никита Андреевич

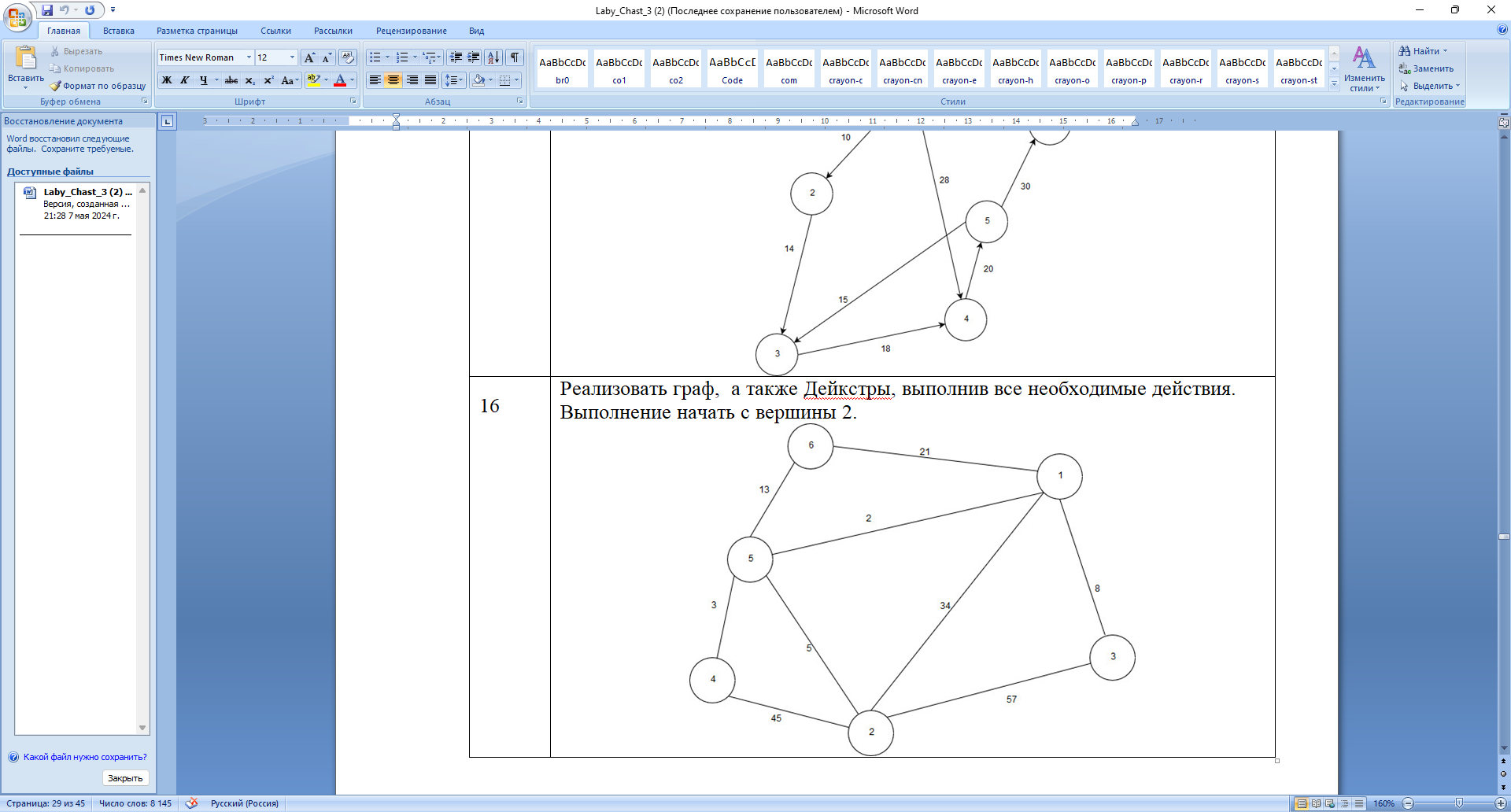
Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь, 2024 г.

**«Графы»**

**//105**

**Задача: 1.**

****

**Анализ задачи:**

1)Для решения данной задачи в C++ необходимо сначала создать класс для графа, который будет содержать информацию о вершинах, ребрах и их весах. Для этого можно использовать структуру с полями для хранения списка смежности, списка ребер и их весов.

2)Затем необходимо реализовать алгоритм Дейкстры для нахождения кратчайших путей от начальной вершины до всех остальных. Алгоритм Дейкстры основан на поиске кратчайшего пути от начальной вершины до всех остальных вершин в графе.

Алгоритм Дейкстры работает следующим образом:

1.Создается множество вершин, которые еще не были посещены, а также массив для хранения расстояний от начальной вершины до всех остальных.

2.Устанавливаем расстояние от начальной вершины до самой себя равным нулю, а до всех остальных вершин - бесконечность.

3.Пока есть непосещенные вершины, выбираем вершину с минимальным расстоянием и обновляем расстояния до всех смежных вершин.

4.Повторяем пункт 3 до тех пор, пока все вершины не будут посещены.

3)Таким образом, для реализации задачи необходимо создать класс для графа, методы для добавления вершин и ребер, а также метод для выполнения алгоритма Дейкстры. После выполнения алгоритма можно получить кратчайшие пути от начальной вершины до всех остальных вершин.

**Код на языке С++:**

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

template <class T>

class Graph

{

public:

vector<T> vertexList;//Вектор вершин

vector < vector<int>> adjMatrix;//Матрица смежности

int maxSize;//Размер вектора вершин и матрицы смежности

Graph() {

maxSize = 0;

}

inline Graph(const int& size)

{

this->maxSize = size;

this->adjMatrix = vector<vector<T>>(size, vector<T>(size));

for (int i = 0; i < this->maxSize; ++i)

{

for (int j = 0; j < this->maxSize; ++j)

{

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

};

};

};

inline bool isFull()

{

return this->vertexList.size() == this->maxSize;

};

bool isEmpty()

{

return (this->vertexList.size() == 0);

}

inline void insertVertex(const T& vert)

{

if (this->isFull())

{

cout << "Невозможно добвать вершину." << endl;

return;

};

this->vertexList.push\_back(vert);

};

inline int GetVertPos(const T& v)

{

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

{

if (this->vertexList[i] == v)

{

return i;

}

}

return -1;

};

inline int GetAmountVerts()

{

return this->vertexList.size();

};

inline int GetWeight(const T& v1, const T& v2)

{

if (this->isEmpty())

{

return 0;

};

int v1\_p = this->GetVertPos(v1);

int v2\_p = this->GetVertPos(v2);

if (v1\_p == -1 || v2\_p == -1)

{

cout << "Одного из узлов в графе не существует." << endl;

return 0;

};

return this->adjMatrix[v1\_p][v2\_p];

};

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

std::vector<T> nbrsList; // создание списка соседей

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

if (pos != -1)

{ /\* проверка, что vertex есть в матрице смежности \*/

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

{

if (this->adjMatrix[pos][i] != 0)

{

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

}

}

} return nbrsList; // возврат списка соседей

};

void InsertEdge(const T& vertex1, const T& vertex2, int weight = 1) {

if (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 << "Ребро между вершинами уже есть" << endl;

return;

}

else

{

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

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

}

}

else

{

cout << "Обеих вершин (или одной из них) нет в графе " << endl;

return;

}

};

void Print() {

if (!(this->isEmpty()))

{

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

printf("%s"," ");

printf("%c",' ');

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

{

printf("%5d ", vertexList[i]);

}

cout << endl;

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

{

printf("%5d", vertexList[i]);

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

printf("%5d ", this->adjMatrix[i][j]);

printf("\n");

}

}

else {

cout << "Граф пуст " << endl;

}

}

int GetAmountEdges()

{

int amount = 0;

if (!this->IsEmpty())

{

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

{

for (int j = 0; j < this->vertList.size(); ++j)

{

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

{

amount++;

}

}

}

}

return amount / 2;

}

void algorithm(int& minindex, int& min, int amountVerts, vector<T>& verh, vector<T>& minSize)

{

do {

minindex = 10000;

min = 10000;

int temp;

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

{ // Если вершину ещё не обошли и вес меньше min

if ((verh[i] == 1) && (minSize[i] < min))

{ // Переприсваиваем значения

min = minSize[i];

minindex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minindex != 10000)

{

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

{

if (this->adjMatrix[minindex][i] > 0)

{

temp = min + this->adjMatrix[minindex][i];

if (temp < minSize[i])

{

minSize[i] = temp;

}

}

}

verh[minindex] = 0;

}

}

while (minindex < 10000);

}

int RestorationPath(vector<int>& ver, int begin\_index, vector<int>& minSize, int amountVerts) {

int end;// индекс конечной вершины

cout << endl << "Введите конец:";

cin >> end;

end -= 1;// индекс конечной вершины = end - 1

ver[0] = end + 1; // начальный элемент - конечная вершина

int k = 1; // индекс предыдущей вершины

int weight = minSize[end]; // вес конечной вершины

while (end != begin\_index) // пока не дошли до начальной вершины

{

for (int i = 0; i < amountVerts; i++) // просматриваем все вершины

if (this->adjMatrix[i][end] != 0) // если связь есть

{

int temp = weight - this->adjMatrix[i][end]; // определяем вес пути из предыдущей вершины

if (temp == minSize[i]) // если вес совпал с рассчитанным

{ // значит из этой вершины и был переход

weight = temp; // сохраняем новый вес

end = i; // сохраняем предыдущую вершину

ver[k] = i + 1; // и записываем ее в массив

k++;

}

}

}

return k;

}

};

int main() {

system("chcp 1251");

system("cls");

int amountVerts = 0, amountEdges = 0, vertex = 0, sourceVertex = 0, targetVertex = 0, edgeweight = 0;

cout << "Введите количество вершин графа: ";

cin >> amountVerts;

cout << endl;

cout << "Введите количество ребер графа: ";

cin >> amountEdges;

cout << endl;

Graph<int> graph(amountVerts);

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

cout << "Вершина: ";

cin >> vertex;

graph.insertVertex(vertex);

cout << endl;

}

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

cout << "Исходная вершина: ";

cin >> sourceVertex;

cout << endl;

cout << "Конечная вершина: ";

cin >> targetVertex;

cout << endl;

cout << "Bec ребра: ";

cin >> edgeweight;

cout << endl;

graph.InsertEdge(sourceVertex, targetVertex, edgeweight);

}

cout << endl;

graph.Print();

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

vector<int> minSize(amountVerts); // минимальное расстояние d

vector<int> verh(amountVerts); // посещенные вершины v

vector<int> ver(amountVerts); // массив посещенных вершин

int temp = 0, minindex = 10000, min = 10000;

int begin\_index = 0;

int end = 0; // индекс конечной вершины

//Инициализация вершин и расстояний

cout << "1";

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

{

minSize[i] = 10000;

verh[i] = 1;

}

cout << "2";

minSize[begin\_index] = 0;

// Шаг алгоритма

cout << "3";

graph.algorithm(minindex, min, amountVerts, verh, minSize);

// Вывод кратчайших расстояний до вершин

cout << "4";

printf("\nКратчайшие расстояния до вершин: \n");

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

printf("%5d ", minSize[i]);

// Восстановление пути

int k = graph.RestorationPath(ver, begin\_index, minSize, amountVerts);

printf("\nВывод кратчайшего пути\n");

for (int i = k - 1; i >= 0; i--)

printf("%3d ", ver[i]);

return 0;

}

**Код OpenGL:**

#pragma once

#include <GL/glut.h>

#include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int n;

int\*\* help;

int\* result;

int\*\*\* mat;

int WinW;

int WinH;

const int maxSize = 20;

int amountVerts;

template<class T>

class Graph

{

private:

vector<T> vertList;

vector<T> labelList;

bool\* visitVerts = 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 GetVerPos(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;

struct vertCoord

{

int x, y;

};

vertCoord vertC[20];

Graph<int> graph;

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] && hmin > \*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 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]);

}

}

}

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

{

preparation(mat, n, help, result);

int s = 0;

answer(mat, n, help, result);

cout << endl << "Отрезки 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 << endl << "Кратчайший путь: ";

int tmp = 0;

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

{

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

{

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

{

if (tmp == 0)

{

cout << i + 1;

}

j = result[i];

tmp = j + 1;

if (tmp > 0)

{

cout << " -> " << tmp;

}

l++;

}

}

}

cout << endl << "Минимальное расстояние: " << s;

cout << endl;

}

template<class T>

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

{

vector<T> nbrsList;

int pos = this->GetVerPos(vertex);

if (pos != (-1))

{

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

{

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

{

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

}

}

}

return nbrsList;

}

template<class T>

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

{

if (!this->IsFull())

{

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

}

else

{

cout << "Граф уже заполнен. Невозможно добавить новую вершину." << endl;

return;

}

}

template<class T>

void Graph<T>::DeleteVertex()

{

this->vertList.pop\_back();

}

template<class T>

int Graph<T>::GetAmountEdges()

{

int amount = 0;

if (!this->IsEmpty())

{

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)

{

amount++;

}

}

}

return amount / 2;

}

else

return 0;

}

template<class T>

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

{

if (!this->IsEmpty())

{

int vert\_1 = this->GetVertPos(vertex1);

int vert\_2 = this->GetVertPos(vertex2);

return adjMatrix[vert\_1][vert\_2];

}

return 0;

}

template<class T>

int Graph<T>::GetAmountVerts()

{

return this->vertList.size();

}

template<class T>

bool Graph<T>::IsEmpty()

{

return this->vertList.size() == 0;

}

template<class T>

bool Graph<T>::IsFull()

{

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

}

template<class T>

int Graph<T>::GetVerPos(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()

{

}

Graph<int> makeGraph()

{

Graph<int> graph;

int amountEdges, sourceVertex, targetVertex, edgeWeight, vertex;

cout << "Количество вершин: ";

cin >> amountVerts;

cout << endl;

cout << "Количество ребер: ";

cin >> amountEdges;

cout << endl;

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

{

int\* vertPtr = &i;

graph.InsertVertex(\*vertPtr);

}

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

{

cout << "Начальная вершина: ";

cin >> sourceVertex;

cout << endl;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина: ";

cin >> targetVertex;

cout << endl;

int\* targetVertPtr = &targetVertex;

cout << "Вес: ";

cin >> edgeWeight;

cout << endl;

graph.InsertEdge(sourceVertex, targetVertex, edgeWeight);

}

cout << endl;

return graph;

}

template<class T>

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

{

if (this->GetVerPos(vertex1) != (-1) && this->GetVerPos(vertex2) != (-1))

{

int vertPos1 = GetVerPos(vertex1);

int vertPos2 = GetVerPos(vertex2);

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

{

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

return;

}

else

{

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

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

}

}

else

{

cout << "Обеих вершин (или одной из них) нет в графе" << endl;

return;

}

}

template<class T>

void Graph<T>::Print()

{

int vertListSize = this->vertList.size();

if (!this->IsEmpty())

{

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

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

{

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

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

{

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

}

cout << endl;

}

}

else

{

cout << "Граф пуст" << endl;

}

}

void setCord(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(1.0, 1.0, 0.63);

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\* front = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(nom);

glRasterPos2i(x1 - 5, y1 - 5);

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

{

glutBitmapCharacter(front, s[j]);

}

}

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);

}

}

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);

}

template<class T>

void Graph<T>::DrawGraph()

{

int n = vertList.size();

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

{

setCord(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 = 60;

int height = 730;

glColor3d(0.0, 0.1, 1.0);

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(0.4, 0.0, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 30);

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

glVertex2i(shift + 135, height - shift);

glVertex2i(shift, height - shift);

glEnd();

glColor3d(1, 1, 1);

drawMenuText("Insert Vertex", shift, height - shift - 2);

glColor3d(0.4, 0.0, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 70);

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

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

glVertex2i(shift, height - shift - 40);

glEnd();

glColor3d(1, 1, 1);

drawMenuText("Delete Vertex", shift, height - shift - 42);

glColor3d(0.4, 0.0, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 110);

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

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

glVertex2i(shift, height - shift - 80);

glEnd();

glColor3d(1, 1, 1);

drawMenuText("Matrix", shift, height - shift - 82);

glColor3d(0.4, 0.0, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 150);

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

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

glVertex2i(shift, height - shift - 120);

glEnd();

glColor3d(1, 1, 1);

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

glColor3d(0.4, 0.0, 0.3);

glBegin(GL\_QUADS);

glVertex2i(shift, height - shift - 190);

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

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

glVertex2i(shift, height - shift - 160);

glEnd();

glColor3d(1, 1, 1);

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

}

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(sourceVertex, targetVertex, edgeWeight);

}

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

{

int sourceVertex;

int targetVertex;

int edgeWeight;

cout << "Введите вершину: ";

cin >> sourceVertex;

cout << endl;

int\* sourceVertPtr = &sourceVertex;

if (sourceVertex == amountVerts)

{

amountVerts--;

graph.DeleteVertex();

}

else

{

cout << "Невозможно удалить вершину!";

}

}

if (x > shift && x<shift + 135 && y>shift + 80 && y < shift + 110)

{

graph.Print();

}

if (x > shift && x<shift + 135 && y>shift + 120 && y < shift + 150)

{

tsalesman(mat, n, help, result);

}

if (x > shift && x<shift + 135 && y>shift + 160 && y < shift + 190)

{

graph = makeGraph();

}

}

glutPostRedisplay();

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.DrawGraph();

drawMenu();

glutSwapBuffers();

}

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);

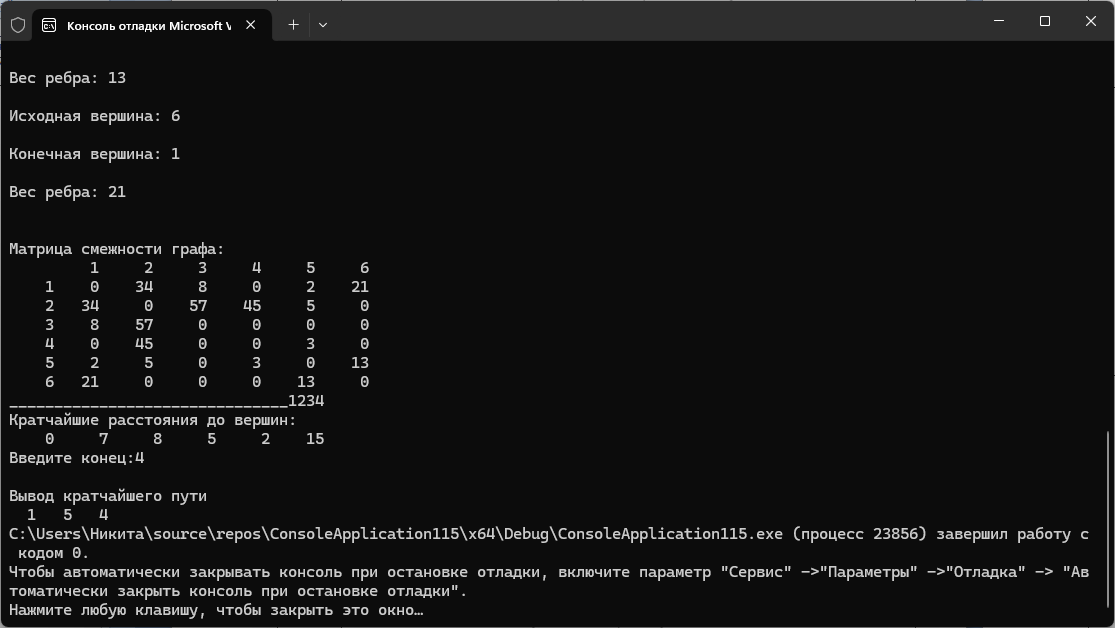
glutMouseFunc(mouseClick);

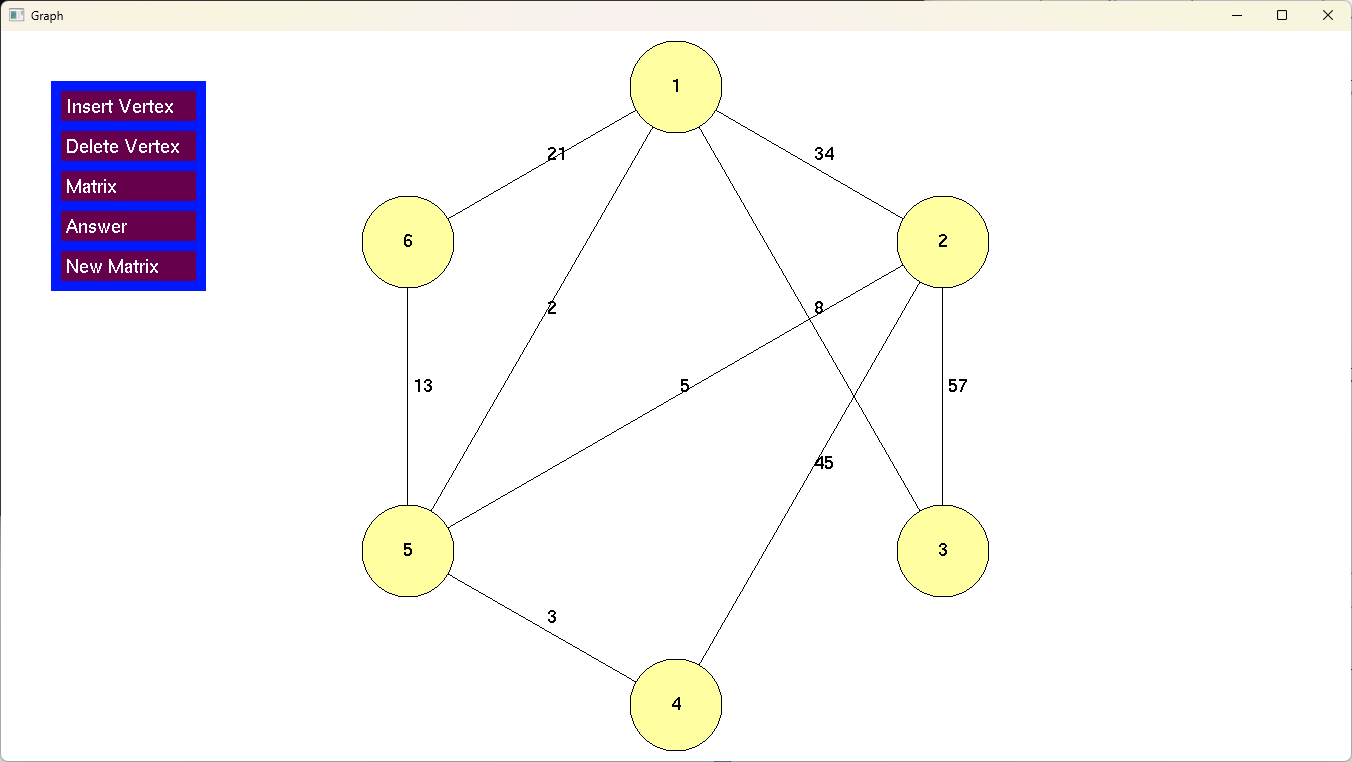
glutMainLoop();

return 0;

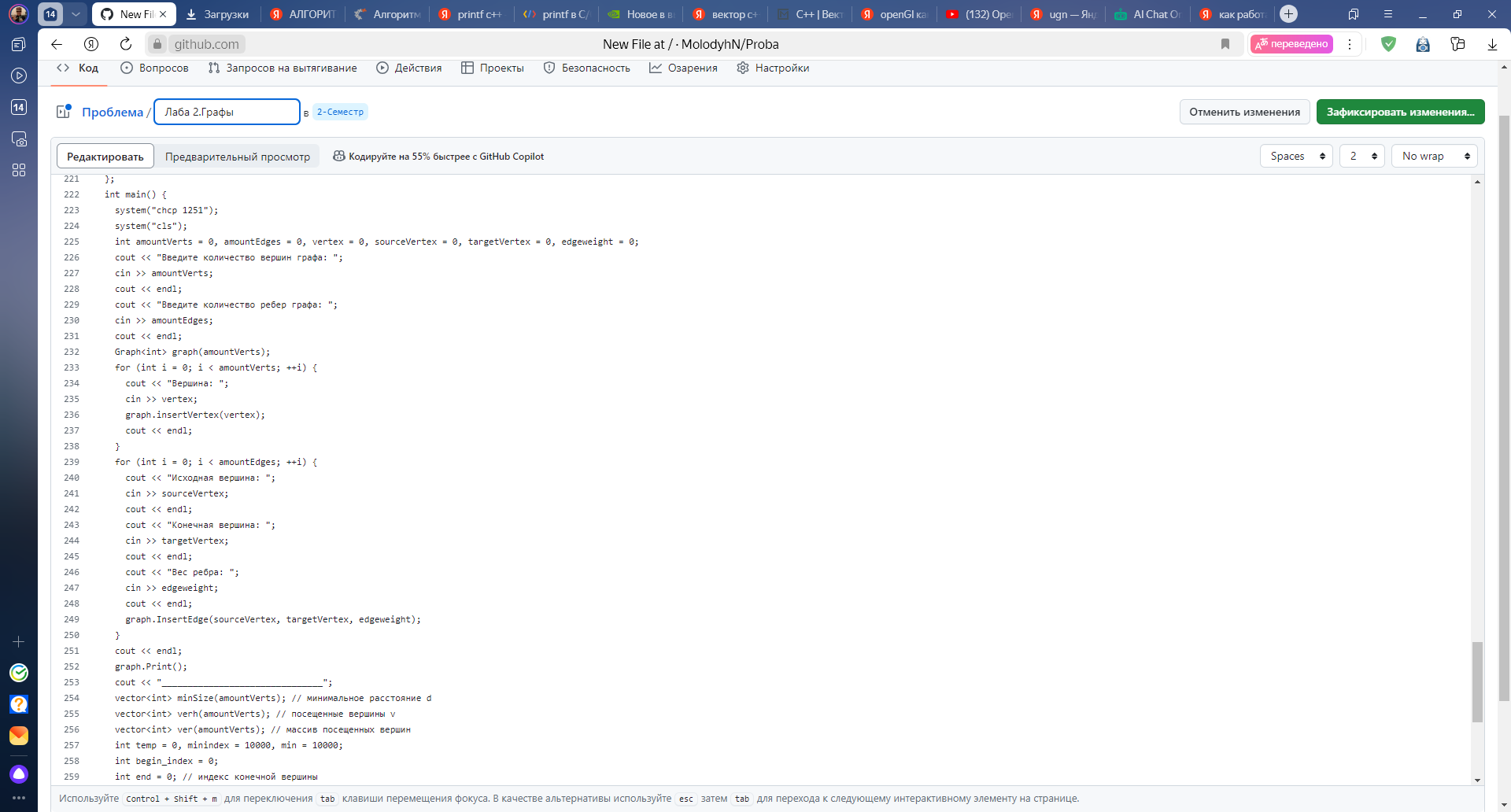
}

**Работа программы:**

****

**OpenGL:**

**Cкрины из гита:**

****

**Ссылка на гит:**

https://github.com/MolodyhN/proba

UML:

