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

ПНИПУ

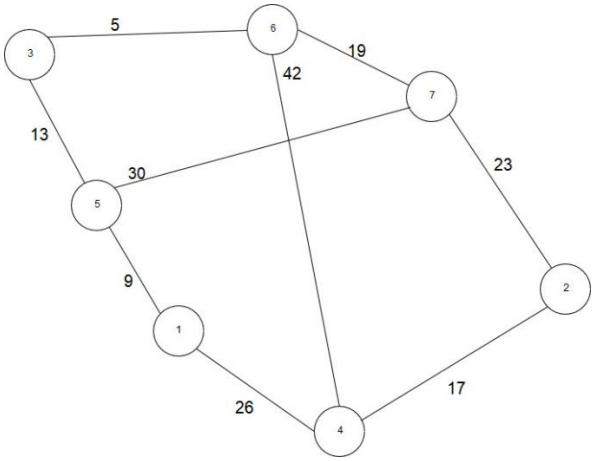
Лабораторная работа
«Графы»

Выполнила:
студентка группы ИВТ-23-2б
Соловьева Екатерина Александровна

Проверила:
доцент кафедры ИТАС
О.А. Полякова

Пермь, 2024 г.

Реализовать граф, а также алгоритм Флойда, построив все необходимые матрицы. Выполнение начать с вершины 7.



Программный код

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <sstream>
#include <Windows.h>
#include <GL\glew.h>
#include <GL\freeglut.h>
#include <iostream>

using namespace std;

int n;
int** help;
int* result;
int*** mat;
int R;
int WinW;
int WinH;
const int maxSize = 20;
int amountVerts=0;

struct vertCoord//Структура установки координат
```

```
{
	int x, y;
};
vertCoord vertC[20];

template<class T>
class Graph
{
	vector<T> vetrexList;
	vector<T> labelList;
	int size;
	bool* visitedVerts = new bool[vetrexList.size()];
public:
	vector<vector<int>> adjMatrix;
	Graph();
	Graph<T>(const int& ksize);
	~Graph();
	void DrawGraph();
	void InsertEdge(const T& vertex1, const T& vertex2, int weight); //Шаблон графа, здесь написаны прототипы функций
	inline void insertVertex(const T& vertex);
	void removeVertex(const T& vertex);
	inline int GetVertPos(const T& vertex);
	inline bool isEmpty();
	inline bool IsFull();
	inline int GetAmountVerts();
	int GetAmountEdges();
	inline int GetWeight(const T& vertex1, const T& vertex2);
	vector<T> GetNbrs(const T& vertex);
	void PrintMatrix();
	void removeEdge(const T& vertex1, const T& vertex2);
	void editEdgeWeight(const T& vertex1, const T& vertex2, int newWeight);
};

Graph<int> graph(20);
template<class T>
vector<T> Graph<T> :: GetNbrs(const T& vetrex) {//Функция для получения вектора соседей
```

```
		vector<T> nbrsList;
		int pos = this->GetVertPos(vetrex);
		if (pos != -1) {
			for (int i = 0; i < this->vetrexList.size(); i++) {
				if (this->adjMatrix[pos][i] != 0) {
					nbrsList.push_back(this->vetrexList[i]);
				}
			}
		}
		return nbrsList;
	}



template<class T>
inline void Graph<T>:: insertVertex(const T& vert) {//Функция, которая добавляет вершину
	if (this->IsFull()) {
		cout << "Невозможно добавить вершину." << endl;
		return;
	}
	this->vetrexList.push_back(vert);
}
template<class T>
void Graph<T>::removeEdge(const T& vertex1, const T& vertex2) {//Функция, которая удаляет ребро
	int pos1 = GetVertPos(vertex1);
	int pos2 = GetVertPos(vertex2);

	if (pos1 == -1 || pos2 == -1) {
		cout << "Одной из вершин нет в графе." << endl;
		return;
	}

	if (adjMatrix[pos1][pos2] == 0) {
		cout << "Ребра между вершинами " << vertex1 << " и " << vertex2 << " нет." << endl;
		return;
	}
```

```
	adjMatrix[pos1][pos2] = 0;
	adjMatrix[pos2][pos1] = 0;

	cout << "Ребро между вершинами " << vertex1 << " и " << vertex2 << " удалено." << endl;
}

template<class T>
void Graph<T>::removeVertex(const T& vertex) {////Функция, которая удаляет вершину
	int pos = GetVertPos(vertex);
	if (pos == -1) {
		cout << "Вершины " << vertex << " нет в графе." << endl;
		return;
	}

	for (int i = 0; i < size; i++) {
		if (adjMatrix[pos][i] != 0) removeEdge(vertex, vetrexList[i]);
		if (adjMatrix[i][pos] != 0) removeEdge(vetrexList[i], vertex);
	}

	vetrexList.erase(vetrexList.begin() + pos);

	// Удаляем столбец pos из каждой строки матрицы
	for (int i = 0; i < size; i++) {
		adjMatrix[i].erase(adjMatrix[i].begin() + pos);
	}
	// Удаляем строку pos из матрицы
	adjMatrix.erase(adjMatrix.begin() + pos);

	size--;

	cout << "Вершина " << vertex << " удалена." << endl;
}

template<class T>
int Graph<T>::GetAmountEdges() {//Функция для получения количества ребер для неориентированного графа
	int amount = 0;
	if (!this->isEmpty()) {
```

```
                for (int i = 0; i < this->vetrexList.size(); i++) {
                        for (int j = 0; j < this->vetrexList.size(); j++) {
                                if (this->adjMatrix[i][j] != 0) {
                                        amount++;
                                }
                        }
                }
        }
        return amount / 2;
}

template<class T>
inline int Graph<T>::GetWeight(const T& g1, const T& g2) {//Получение веса между вершинами
        if (this->isEmpty()) {
                return 0;
        }
        int g1_p = this->GetVertPos(g1);
        int g2_p = this->GetVertPos(g2);
        if (g1_p == -1 || g2_p == -1) {
                cout << "Одного из выбранных узлов в графе не существует!";
                return 0;
        }
        return this->adjMatrix[g1_p][g2_p];
}

template<class T>
inline int Graph<T>::GetAmountVerts() {//Получение количества вершин
        return this->vetrexList.size();
}
template<class T>
inline bool Graph<T>::isEmpty() {//Проверка графа на то, что он пуст
        return this->vetrexList.size() == 0;
}
template<class T>//Проверка графа на то, что он заполнен
inline bool Graph<T>::IsFull() {
        return (vetrexList.size() == maxSize);
}
```

```
template <class T>
inline int Graph<T>::GetVertPos(const T& g) {//Получение индекса вершин
	for (int i = 0; i < vetrexList.size(); i++) {
		if (this->vetrexList[i] == g) {
			return i;
		}
	}
	return -1;
}
template<class T>
Graph<T>::Graph() {
	size = maxSize;
	labelList.resize(size, 1000000);
	adjMatrix.resize(size, vector<int>(size, 0));
	visitedVerts = new bool[size];
}

template<class T>
Graph<T>::Graph(const int& ksize) {
	size = ksize;
	labelList.resize(size, 1000000);
	adjMatrix.resize(size, vector<int>(size, 0));
	visitedVerts = new bool[size];
}

template<class T>
Graph<T>::~Graph() {//Дестркутор графа

}

template<class T>
void Graph<T>::InsertEdge(const T& vetrex1, const T& vetrex2, int weight) {//Вставка ребра для неориентированного графа
	if (GetVertPos(vetrex1) != (-1) && this->GetVertPos(vetrex2) != (-1)) {
		int vertPos1 = GetVertPos(vetrex1);
		int vertPos2 = GetVertPos(vetrex2);
```

```
                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 << "Какой-либо вершины нет в графе" << endl;
                return;
        }
}

template<class T>
void Graph<T>::PrintMatrix() {//Печать матрицы смежности графа
        if (!this->isEmpty()) {
                cout << "Матрица смежности: " << endl;
                cout << "- ";
                for (int i = 0; i < vetrexList.size(); i++) {
                        cout << " " << vetrexList[i] << " ";
                }
                cout << endl;
                for (int i = 0; i < this->vetrexList.size(); i++) {
                        cout << this->vetrexList[i] << " ";
                        for (int j = 0; j < this->vetrexList.size(); j++) {
                                cout << " " << this->adjMatrix[i][j] << " ";
                        }
                        cout << endl;
                }
        }
        else {
                cout << "Граф пуст" << endl;
        }
}
template<class T>
```

```
void Graph<T>::editEdgeWeight(const T& vertex1, const T& vertex2, int newWeight) {//Функция, которая меняет вес ребра между вершинами
	int pos1 = GetVertPos(vertex1);
	int pos2 = GetVertPos(vertex2);

	if (pos1 == -1 || pos2 == -1) {
		cout << "Одной из вершин нет в графе." << endl;
		return;
	}

	if (adjMatrix[pos1][pos2] == 0) {
		cout << "Ребра между вершинами " << vertex1 << " и " << vertex2 << " нет." << endl;
		return;
	}

	adjMatrix[pos1][pos2] = newWeight;
	adjMatrix[pos2][pos1] = newWeight;

	cout << "Вес ребра между вершинами " << vertex1 << " и " << vertex2 << " изменен на " << newWeight << "." << endl;
}

void answer(int*** mat, int n, int** help, int* path)//Эта функция реализует алгоритм решения задачи коммивояжера, используя Венгерский алгоритм.
{
	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)// Эта функция подготавливает данные для алгоритма TSP (коммивояжера)
{
        n = amountVerts;// Присваиваем количество вершин из графа
        // Выделяем память под вспомогательные матрицы help и result
        help = new int* [n];
        result = new int[n];
        // Выделяем память под трехмерную матрицу mat, которая будет хранить матрицу смежности графа
        mat = new int** [n];
        // Инициализируем help
        for (int i = 0; i <= n; i++)
        {
                help[i] = new int[n];
        }
        // Заполняем mat значениями из матрицы смежности графа
        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 TSP(int*** mat, int n, int** help, int* result)// Эта функция является точкой входа для решения задачи коммивояжера (TSP).

```
{
        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;
}



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)//Функция, предназначенная для рисования круга
```

```
{
	glColor3f(1.0,0.0,0.0);
	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)//Отрисовка текста в вершине
{
	GLvoid* font = GLUT_BITMAP_TIMES_ROMAN_24;
	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)//Отрисовка вершины, текста в ней
{
```

```
	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) {//Отрисовка ребра, и текста на ребре
	glColor3f(0.0, 0.0, 0.0);
	glBegin(GL_LINES);
	glVertex2i(x0, y0);
	glVertex2i(x1, y1);
	glEnd();

	glColor4f(1.0f, 1.0f, 1.0f, 0.0f);
	drawText(text, (x0 + x1) / 2 + 11, (y0 + y1) / 2 + 11);
}



template<class T>
void Graph<T>::DrawGraph()//Главная функция, которая рисует сам граф
{
	int n = vetrexList.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)//Функция отвечающая за изменение размера вершин
{
		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)//Функция для текста и его шрифта в менюшке
{
		GLvoid* font = GLUT_BITMAP_9_BY_15;
		string s = text;
		glRasterPos2i(x1 + 5, y1 - 20);
		for (int j = 0; j < s.length(); j++)
			glutBitmapCharacter(font, s[j]);
}

void drawMenu()//Рисуется меню с соответсвующими функциями
{
		int shift = 60;
		int height = 730;

		glColor3d(0.0, 0.0, 0.0);
		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("insertVertex", shift, height - shift - 2);

glColor3d(0.0, 0.0, 0.0);
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("DeleteVertex", shift, height - shift - 42);

glColor3d(0.0, 0.0, 0.0);
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("PrintMatrix", shift, height - shift - 82);

glColor3d(0.0, 0.0, 0.0);
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("TSP", shift, height - shift - 122);

glColor3d(0.0, 0.0, 0.0);
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("editEdgeWeight", shift, height - shift - 162);
}

void mouseClick(int btn, int stat, int x, int y) {//Функция, которая позваляет взаимодействовать с кодом через визуализацию, изменять, удалять и т.д.
        int shift = 60;
        int height = 730;

        if (stat == GLUT_DOWN) {

                if (x > shift && x < shift + 135 && y >  shift && y < shift + 30)
                {
                        int vertex;
                        int sourceVertex;
                        int targetVetrex;
                        int edgeWeight;
                        int Weight;
                        int g, k;
                        cout << "Введите кол-во вершин, которые вы хотите добавить: ";
                        cin >> g;
                        cout << "Введите кол-во ребёр, которые хотите добавить: ";
                        cin >> k;
                        for (int i = 0; i < g; i++) {
                                cout << "Вершина: ";
                                cin >> vertex;
                                graph.insertVertex(vertex);
                                amountVerts++;
                                cout << endl;
                        }
                        for (int i = 0; i < k; i++) {
                                cout << "Исходная вершина: ";
                                cin >> sourceVertex;
                                cout << endl;
                                cout << "Конечная вершина: ";
```

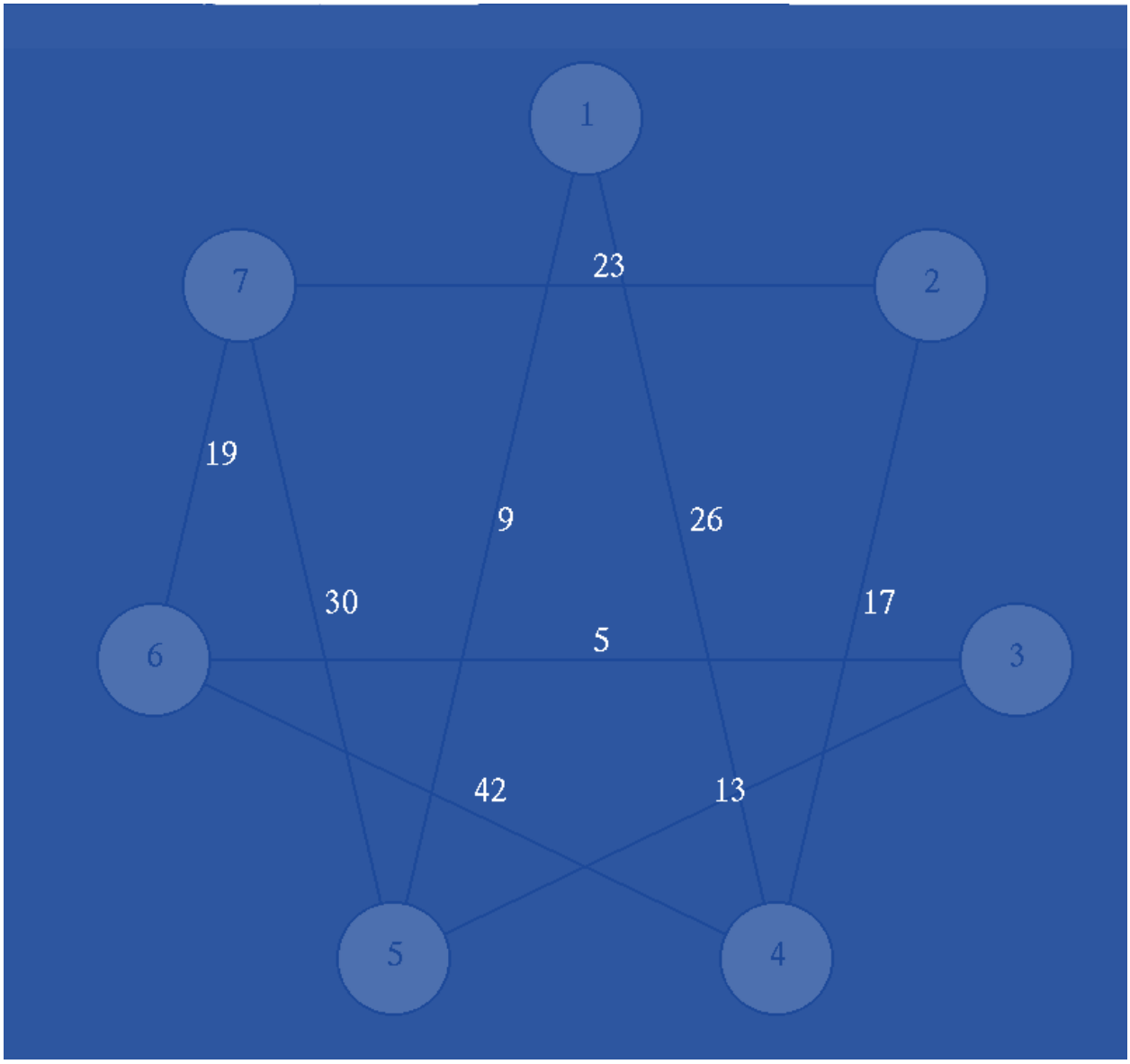
```
                cin >> targetVetrex;
                cout << endl;
                cout << "Вес ребра: ";
                cin >> Weight;
                cout << endl;
                int* targetVerPtr = &targetVetrex;
                graph.InsertEdge(sourceVertex, targetVetrex, Weight);
        }


}
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;

                graph.removeVertex(sourceVertex);
                amountVerts--;



}
if (x > shift && x < shift + 135 && y >  shift + 80 && y < shift + 100)
{
        graph.PrintMatrix();
}
if (x > shift && x < shift + 135 && y >  shift + 120 && y < shift + 140)
{
        TSP(mat, n, help, result);
}
if (x > shift && x < shift + 135 && y >  shift + 160 && y < shift + 180)
{
        int vertex,  Weight, vertex1;
        cout << "Введите номера вершин, между которыми нужно изменить вес ребра: ";
        cin >> vertex;
```

```
                cin >> vertex1;
                cout << endl << endl;
                cout << "Введите нужный вес: ";
                cin >> Weight;
                graph.editEdgeWeight(vertex, vertex1, Weight);
            }
        }
        glutPostRedisplay();
}

void display()//Фунция вызова экрана и вызова функции отрисовки графа
{
        glShadeModel(GL_SMOOTH);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, WinW, 0, WinH);
        glViewport(0, 0, WinW, WinH);
        glClearColor(0.0, 0.0, 1.0, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        graph.DrawGraph();
        drawMenu();
        glutSwapBuffers();
}

int main(int argc, char* argv[])
{
        setlocale(LC_ALL, "rus");
        system("chcp 1251>NULL");
        glutInit(&argc, argv);
        int Verts, Edges, vertex, sourceVertex, targetVetrex, Weight;
        cout << "Введите количество вершин: " << endl;
        cin >> Verts;
        cout << "Введите количество ребер графа: " << endl;
        cin >> Edges;
        cout << endl;
        for (int i = 0; i < Verts; i++) {
            cout << "Вершина: ";
```

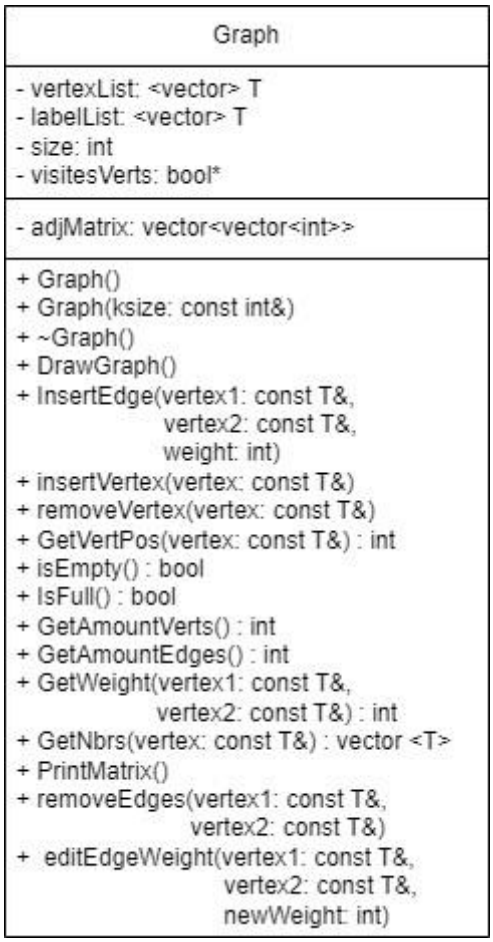
```
            cin >> vertex;
            graph.insertVertex(vertex);
            amountVerts++;
            cout << endl;
        }
        for (int i = 0; i < Edges; i++) {
            cout << "Исходная вершина: ";
            cin >> sourceVertex;
            cout << endl;
            cout << "Конечная вершина: ";
            cin >> targetVetrex;
            cout << endl;
            cout << "Вес ребра: ";
            cin >> Weight;
            cout << endl;
            int* targetVerPtr = &targetVetrex;
            graph.InsertEdge(sourceVertex, targetVetrex, Weight);
        }
        cout << endl;
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowSize(1350, 730);
        glutCreateWindow("Graph");
        WinW = glutGet(GLUT_WINDOW_WIDTH);
        WinH = glutGet(GLUT_WINDOW_HEIGHT);
        glLineWidth(2);
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMouseFunc(mouseClick);
        glutMainLoop();
        return 0;
}
```

Работа программы

```
Введите количество вершин:
7
Введите количество ребер графа:
9

Вершина: 1

Вершина: 2

Вершина: 3

Вершина: 4

Вершина: 5

Вершина: 6

Вершина: 7

Исходная вершина: 7

Конечная вершина: 6

Вес ребра: 19

Исходная вершина: 7

Конечная вершина: 5

Вес ребра: 30

Исходная вершина: 7

Конечная вершина: 2

Вес ребра: 23

Исходная вершина: 6

Конечная вершина: 3

Вес ребра: 5

Исходная вершина: 6

Конечная вершина: 4

Вес ребра: 42

Исходная вершина: 5

Конечная вершина: 3

Вес ребра: 13

Исходная вершина: 5

Конечная вершина: 1

Вес ребра: 9

Исходная вершина: 2

Конечная вершина: 4

Вес ребра: 17

Исходная вершина: 4

Конечная вершина: 1

Вес ребра: 26


Матрица смежности:
-  1  2  3  4  5  6  7
1  0  0  0  26  9  0  0
2  0  0  0  17  0  0  23
3  0  0  0  0  13  5  0
4  26  17  0  0  0  42  0
5  9  0  13  0  0  0  30
6  0  0  5  42  0  0  19
7  0  23  0  0  30  19  0

Отрезки путей: 1 -> 5   2 -> 4  3 -> 6  4 -> 1  5 -> 3  6 -> 7  7 -> 2

Кратчайший путь: 1 -> 5 -> 3 -> 6 -> 7 -> 2 -> 4 -> 1
Минимальное расстояние: 112
```


1
7
23
2
19
9
26
30
17
6
5
3
42
13
5
4

UML


Graph
- vertexList: <vector> T
- labelList: <vector> T
- size: int
- visitesVerts: bool*
- adjMatrix: vector<vector<int>>
+ Graph()
+ Graph(ksize: const int&)
+ ~Graph()
+ DrawGraph()
+ InsertEdge(vertex1: const T&,
vertex2: const T&,
weight: int)
+ insertVertex(vertex: const T&)
+ removeVertex(vertex: const T&)
+ GetVertPos(vertex: const T&) : int
+ isEmpty() : bool
+ IsFull() : bool
+ GetAmountVerts() : int
+ GetAmountEdges() : int
+ GetWeight(vertex1: const T&,
vertex2: const T&) : int
+ GetNbrs(vertex: const T&) : vector <T>
+ PrintMatrix()
+ removeEdges(vertex1: const T&,
vertex2: const T&)
+ editEdgeWeight(vertex1: const T&,
vertex2: const T&,
newWeight: int)