Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

ОТЧЁТ

Тема: «Графы»

Выполнили

Студенты РИС-24-1б Конькова С. С.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2025

**Постановка задачи**

Требуется реализовать работу с графом.  
1.3 Реализовать Алгоритмы на  
C++:  
1)Обход в ширину.  
2)Обход в глубину.  
3)Алгоритм Дейкстры.  
  
1.4 Требования:  
1) Пользовательский интерфейс - инструменты по решению разработчика (Windows Forms, например)  
2) Визуализация графа с использованием любой доступной графической библиотеки – SFML(Предпочтительно), SDL, OpenGL ( пример реализации представлен в файле: «Визуализация деревьев.docx»)  
3) Реализованные алгоритмы должны справляться не только с графом, представленным автором, а также применяться к другим  
Графам.  
4) Необходимы функции для редактирования графа: Создание(добавление) и удаление вершины и ребра. Редактирование весов ребер. Редактирование матрицы смежности (или инцидентности - в зависимости от реализации).

**Анализ**

1. Создается главное окно (GraphWidget). Инициализируется графическая сцена и представление
2. Настраиваются элементы управления:
   * Поля ввода для меток вершин и параметров ребер
   * Кнопки для добавления элементов и выполнения алгоритмов
3. Создан класс для вершин графа с обработкой перемещения
4. Реализован основной класс GraphWidget
5. Добавлены контейнеры для хранения вершин и рёбер и матрица смежности
6. Реализованы методы добавления вершин (с проверкой дубликатов) и рёбер (с учётом радиуса вершин)
7. Настроено автоматическое обновление позиций рёбер при перемещении вершин
8. Добавлены UI-элементы для управления графом
9. Реализованы алгоритмы: Дейкстры (кратчайшие пути), BFS и DFS (обходы графа)
10. Настроен вывод результатов
11. В main() создан QApplication и запущено главное окно

**Код.**

#include <QApplication>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsEllipseItem>

#include <QGraphicsTextItem>

#include <QGraphicsLineItem>

#include <QPushButton>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QLineEdit>

#include <QWidget>

#include <QMessageBox>

#include <QGraphicsItem>

#include <QVariant>

#include <vector>

#include <queue>

#include <stack>

#include <limits>

#include <cmath>

using namespace std;

class GraphWidget;

class VertexItem : public QGraphicsEllipseItem {

public:

GraphWidget\* graph;

int index;

QGraphicsTextItem\* label;

VertexItem(int i, GraphWidget\* g)

: QGraphicsEllipseItem(-20, -20, 40, 40), graph(g), index(i), label(nullptr) {

setBrush(QColor(255, 182, 193)); // Розовый цвет вершин

setFlags(ItemIsMovable | ItemSendsGeometryChanges);

}

protected:

QVariant itemChange(GraphicsItemChange change, const QVariant &value) override;

};

class GraphWidget : public QWidget {

Q\_OBJECT

QGraphicsScene\* scene;

QGraphicsView\* view;

QLineEdit \*inputVertex, \*inputEdgeStart, \*inputEdgeEnd, \*inputEdgeWeight, \*inputStartVertex;

QPushButton \*btnAddVertex, \*btnAddEdge, \*btnDijkstra, \*btnBFS, \*btnDFS;

struct EdgeItem {

QGraphicsLineItem\* line;

QGraphicsTextItem\* label;

int startIndex, endIndex, weight;

};

vector<VertexItem\*> vertices;

vector<EdgeItem> edges;

vector<vector<int>> adjMatrix;

int maxSize = 20;

public:

GraphWidget(QWidget\* parent = nullptr) : QWidget(parent) {

adjMatrix.resize(maxSize, vector<int>(maxSize, 0));

QVBoxLayout\* mainLayout = new QVBoxLayout(this);

view = new QGraphicsView(this);

scene = new QGraphicsScene(this);

view->setScene(scene);

view->setRenderHint(QPainter::Antialiasing);

mainLayout->addWidget(view);

QHBoxLayout\* layout1 = new QHBoxLayout;

inputVertex = new QLineEdit; inputVertex->setPlaceholderText("Vertex label");

btnAddVertex = new QPushButton("Add Vertex");

layout1->addWidget(inputVertex);

layout1->addWidget(btnAddVertex);

QHBoxLayout\* layout2 = new QHBoxLayout;

inputEdgeStart = new QLineEdit; inputEdgeStart->setPlaceholderText("Start");

inputEdgeEnd = new QLineEdit; inputEdgeEnd->setPlaceholderText("End");

inputEdgeWeight = new QLineEdit; inputEdgeWeight->setPlaceholderText("Weight");

btnAddEdge = new QPushButton("Add Edge");

layout2->addWidget(inputEdgeStart);

layout2->addWidget(inputEdgeEnd);

layout2->addWidget(inputEdgeWeight);

layout2->addWidget(btnAddEdge);

QHBoxLayout\* layout3 = new QHBoxLayout;

inputStartVertex = new QLineEdit; inputStartVertex->setPlaceholderText("Start Vertex");

btnDijkstra = new QPushButton("Dijkstra");

btnBFS = new QPushButton("BFS");

btnDFS = new QPushButton("DFS");

layout3->addWidget(inputStartVertex);

layout3->addWidget(btnDijkstra);

layout3->addWidget(btnBFS);

layout3->addWidget(btnDFS);

mainLayout->addLayout(layout1);

mainLayout->addLayout(layout2);

mainLayout->addLayout(layout3);

connect(btnAddVertex, &QPushButton::clicked, this, &GraphWidget::onAddVertex);

connect(btnAddEdge, &QPushButton::clicked, this, &GraphWidget::onAddEdge);

connect(btnDijkstra, &QPushButton::clicked, this, &GraphWidget::onDijkstra);

connect(btnBFS, &QPushButton::clicked, this, &GraphWidget::onBFS);

connect(btnDFS, &QPushButton::clicked, this, &GraphWidget::onDFS);

}

void updateEdges(int idx) {

if (idx < 0 || idx >= (int)vertices.size()) return;

auto\* vertex = vertices[idx];

QPointF center = vertex->scenePos();

// Центрировать текст внутри круга

if (vertex->label) {

QRectF br = vertex->label->boundingRect();

vertex->label->setPos(center.x() - br.width() / 2,

center.y() - br.height() / 2);

}

// Обновить линии и подписи ребер, связанных с этой вершиной

for (auto& edge : edges) {

if (edge.startIndex == idx || edge.endIndex == idx) {

QPointF p1 = vertices[edge.startIndex]->scenePos();

QPointF p2 = vertices[edge.endIndex]->scenePos();

QLineF fullLine(p1, p2);

const qreal radius = 20.0;

if (fullLine.length() > 2 \* radius) {

fullLine.setP1(fullLine.pointAt(radius / fullLine.length()));

fullLine.setP2(fullLine.pointAt(1 - radius / fullLine.length()));

}

edge.line->setLine(fullLine);

// Позиционирование текста веса ребра перпендикулярно ребру

QPointF mid = (fullLine.p1() + fullLine.p2()) / 2;

QPointF normal = QPointF(-(fullLine.p2().y() - fullLine.p1().y()),

fullLine.p2().x() - fullLine.p1().x());

qreal length = sqrt(normal.x() \* normal.x() + normal.y() \* normal.y());

if (length > 0) {

normal /= length;

normal \*= 15; // Отступ от ребра

}

edge.label->setPos(mid + normal - QPointF(edge.label->boundingRect().width()/2,

edge.label->boundingRect().height()/2));

}

}

}

private slots:

void onAddVertex() {

bool ok;

int label = inputVertex->text().toInt(&ok);

if (!ok) {

QMessageBox::warning(this, "Ошибка", "Введите корректный номер вершины (целое число)");

return;

}

// Проверка на дублирование метки вершины

for (auto\* v : vertices) {

if (v->label->toPlainText().toInt() == label) {

QMessageBox::warning(this, "Ошибка", "Вершина с таким номером уже существует");

return;

}

}

if ((int)vertices.size() >= maxSize) {

QMessageBox::warning(this, "Ошибка", "Превышено максимальное число вершин");

return;

}

int index = (int)vertices.size();

int r = 150;

double angle = (2 \* M\_PI \* index) / maxSize;

QPointF pos(250 + r \* cos(angle), 250 + r \* sin(angle));

auto\* vertex = new VertexItem(index, this);

vertex->setPos(pos);

scene->addItem(vertex);

auto\* text = scene->addText(QString::number(label));

text->setDefaultTextColor(Qt::black);

text->setZValue(1);

QRectF br = text->boundingRect();

text->setPos(pos.x() - br.width() / 2, pos.y() - br.height() / 2);

vertex->label = text;

vertices.push\_back(vertex);

inputVertex->clear();

}

void onAddEdge() {

bool ok1, ok2, ok3;

int s = inputEdgeStart->text().toInt(&ok1);

int e = inputEdgeEnd->text().toInt(&ok2);

int w = inputEdgeWeight->text().toInt(&ok3);

if (!ok1 || !ok2 || !ok3 || s == e) {

QMessageBox::warning(this, "Ошибка", "Некорректный ввод ребра");

return;

}

int i = findIndex(s);

int j = findIndex(e);

if (i == -1 || j == -1) {

QMessageBox::warning(this, "Ошибка", "Одна из вершин не найдена");

return;

}

if (adjMatrix[i][j] != 0) {

QMessageBox::warning(this, "Ошибка", "Ребро уже существует");

return;

}

adjMatrix[i][j] = adjMatrix[j][i] = w;

QPointF p1 = vertices[i]->scenePos();

QPointF p2 = vertices[j]->scenePos();

QLineF line(p1, p2);

const qreal radius = 20.0;

if (line.length() > 2 \* radius) {

line.setP1(line.pointAt(radius / line.length()));

line.setP2(line.pointAt(1 - radius / line.length()));

}

auto\* lineItem = scene->addLine(line, QPen(Qt::black, 2));

auto\* text = scene->addText(QString::number(w));

text->setDefaultTextColor(Qt::black); // Черный цвет веса ребра

// Позиционирование текста веса ребра перпендикулярно ребру

QPointF mid = (line.p1() + line.p2()) / 2;

QPointF normal = QPointF(-(line.p2().y() - line.p1().y()),

line.p2().x() - line.p1().x());

qreal length = sqrt(normal.x() \* normal.x() + normal.y() \* normal.y());

if (length > 0) {

normal /= length;

normal \*= 15; // Отступ от ребра

}

text->setPos(mid + normal - QPointF(text->boundingRect().width()/2,

text->boundingRect().height()/2));

edges.push\_back({lineItem, text, i, j, w});

inputEdgeStart->clear();

inputEdgeEnd->clear();

inputEdgeWeight->clear();

}

void onDijkstra() {

bool ok;

int label = inputStartVertex->text().toInt(&ok);

int start = findIndex(label);

if (!ok || start == -1) {

QMessageBox::warning(this, "Ошибка", "Вершина не найдена");

return;

}

vector<int> dist(vertices.size(), numeric\_limits<int>::max());

vector<bool> visited(vertices.size(), false);

dist[start] = 0;

for (size\_t i = 0; i < vertices.size(); ++i) {

int u = -1;

for (size\_t j = 0; j < vertices.size(); ++j)

if (!visited[j] && (u == -1 || dist[j] < dist[u])) u = j;

if (dist[u] == numeric\_limits<int>::max()) break;

visited[u] = true;

for (size\_t v = 0; v < vertices.size(); ++v)

if (adjMatrix[u][v] && dist[u] + adjMatrix[u][v] < dist[v])

dist[v] = dist[u] + adjMatrix[u][v];

}

QString result = "Dijkstra:\n";

for (size\_t i = 0; i < vertices.size(); ++i) {

result += QString::number(label) + " → " + vertices[i]->label->toPlainText() + ": "

+ (dist[i] == numeric\_limits<int>::max() ? "∞" : QString::number(dist[i])) + "\n";

}

QMessageBox::information(this, "Результат", result);

}

void onBFS() {

bool ok;

int label = inputStartVertex->text().toInt(&ok);

int start = findIndex(label);

if (!ok || start == -1) {

QMessageBox::warning(this, "Ошибка", "Вершина не найдена");

return;

}

QString res = "BFS: ";

vector<bool> visited(vertices.size(), false);

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int u = q.front(); q.pop();

res += vertices[u]->label->toPlainText() + " ";

for (size\_t v = 0; v < vertices.size(); ++v) {

if (adjMatrix[u][v] && !visited[v]) {

visited[v] = true;

q.push(v);

}

}

}

QMessageBox::information(this, "Обход в ширину", res);

}

void onDFS() {

bool ok;

int label = inputStartVertex->text().toInt(&ok);

int start = findIndex(label);

if (!ok || start == -1) {

QMessageBox::warning(this, "Ошибка", "Вершина не найдена");

return;

}

QString res = "DFS: ";

vector<bool> visited(vertices.size(), false);

stack<int> st;

st.push(start);

while (!st.empty()) {

int u = st.top(); st.pop();

if (!visited[u]) {

visited[u] = true;

res += vertices[u]->label->toPlainText() + " ";

for (int v = (int)vertices.size() - 1; v >= 0; --v) {

if (adjMatrix[u][v] && !visited[v]) st.push(v);

}

}

}

QMessageBox::information(this, "Обход в глубину", res);

}

private:

int findIndex(int label) {

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

if (vertices[i]->label->toPlainText().toInt() == label)

return i;

return -1;

}

};

QVariant VertexItem::itemChange(GraphicsItemChange change, const QVariant &value) {

if (change == ItemPositionChange && graph) {

graph->updateEdges(index);

}

return QGraphicsEllipseItem::itemChange(change, value);

}

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

QApplication a(argc, argv);

GraphWidget w;

w.resize(800, 600);

w.show();

return a.exec();

}

#include "main.moc"

**Результаты выполнения программы.**

Изображение выглядит как диаграмма, текст, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**UML-диаграмма**

