Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Графы

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

Реализовать алгоритмы для собственного варианта графа, имеющего не менее 6 вершин.

Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)

2. Визуализация графа с использованием любой доступной графической библиотеки (SFML, SDL, OpenGL и подобных)

3. Реализованные алгоритмы должны справляться как с графом, представленным в задании варианта, так и с другими на усмотрение проверяющего.

4. Необходимо реализовать функции для редактирования графа:

- Создание новой вершины.

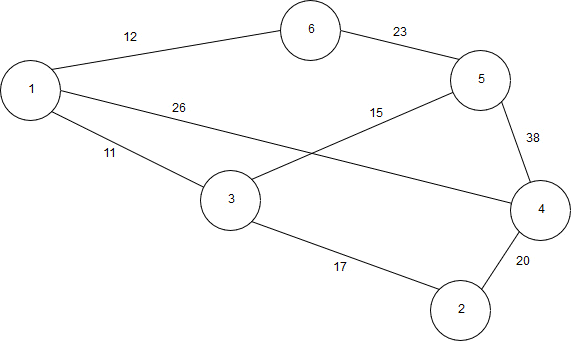
- Удаление вершины.

- Добавление и удаление ребра.

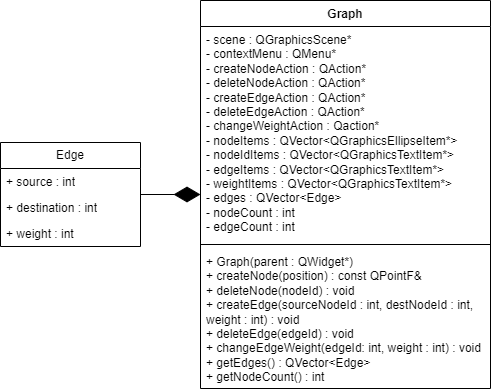
- Редактирование весов ребер.

- Редактирование матрицы смежности (или инцидентности в зависимости от реализации).

Вариант 10:



UML-диаграмма



Код программы

#include <QtWidgets>

#include <queue>

#include <stack>

*struct* **Edge**

{

int source;

int destination;

int weight;

};

*class* **Graph** : *public* QGraphicsView

{

Q\_OBJECT

*public*:

**Graph**(QWidget \*parent = *nullptr*) : QGraphicsView(*parent*), nodeCount(0), edgeCount(0)

{

scene = *new* QGraphicsScene(*this*);

setScene(*scene*);

setRenderHint(QPainter::*Antialiasing*);

setDragMode(*ScrollHandDrag*);

connect(*this*, &Graph::nodeCreated, *this*, &Graph::createNode);

connect(*this*, &Graph::nodeDeleted, *this*, &Graph::deleteNode);

connect(*this*, &Graph::edgeCreated, *this*, &Graph::createEdge);

connect(*this*, &Graph::edgeDeleted, *this*, &Graph::deleteEdge);

connect(*this*, &Graph::edgeWeightChanged, *this*, &Graph::changeEdgeWeight);

contextMenu = *new* QMenu(*this*);

createNodeAction = *new* QAction(tr("Создать вершину"), *this*);

deleteNodeAction = *new* QAction(tr("Удалить вершину"), *this*);

createEdgeAction = *new* QAction(tr("Вставить ребро"), *this*);

deleteEdgeAction = *new* QAction(tr("Удалить ребро"), *this*);

changeWeightAction = *new* QAction(tr("Изменить вес"), *this*);

contextMenu->addAction(*createNodeAction*);

contextMenu->addAction(*deleteNodeAction*);

contextMenu->addAction(*createEdgeAction*);

contextMenu->addAction(*deleteEdgeAction*);

contextMenu->addAction(*changeWeightAction*);

connect(createNodeAction, &QAction::triggered, *this*, &Graph::onCreateNodeActionTriggered);

connect(deleteNodeAction, &QAction::triggered, *this*, &Graph::onDeleteNodeActionTriggered);

connect(createEdgeAction, &QAction::triggered, *this*, &Graph::onCreateEdgeActionTriggered);

connect(deleteEdgeAction, &QAction::triggered, *this*, &Graph::onDeleteEdgeActionTriggered);

connect(changeWeightAction, &QAction::triggered, *this*, &Graph::onChangeWeightActionTriggered);

}

void **createNode**(*const* QPointF& position)

{

QGraphicsEllipseItem\* nodeItem = scene->addEllipse(position.x() - 20, position.y() - 20, 40, 40);

QGraphicsTextItem\* nodeId = scene->addText(QString::number(nodeCount+1));

nodeId->setPos(position.x() - 5, position.y() - 5);

nodeItems.append(nodeItem);

nodeIdItems.append(nodeId);

++nodeCount;

}

void **deleteNode**(int nodeId)

{

*if* (nodeId >= 0 && nodeId < nodeItems.size())

{

scene->removeItem(*nodeItems[nodeId]*);

scene->removeItem(*nodeIdItems[nodeId]*);

*delete* nodeItems[nodeId];

*delete* nodeIdItems[nodeId];

nodeItems.remove(nodeId);

nodeIdItems.remove(nodeId);

*for* (int i = edges.size() - 1; i >= 0; --i) {

*if* (edges[i].source == nodeId || edges[i].destination == nodeId)

{

deleteEdge(i);

}

}

--nodeCount;

}

}

void **createEdge**(int sourceNodeId, int destNodeId, int weight)

{

*if* (sourceNodeId >= 0 && sourceNodeId < nodeCount && destNodeId >= 0 && destNodeId < nodeCount)

{

QPointF sourcePos = nodeIdItems[sourceNodeId]->pos();

QPointF destPos = nodeIdItems[destNodeId]->pos();

QGraphicsLineItem\* lineItem = *new* QGraphicsLineItem(sourcePos.x(),

sourcePos.y(),

destPos.x(),

destPos.y());

lineItem->setPen(QPen(Qt::*black*, 1));

scene->addItem(*lineItem*);

QPointF centerPos = (sourcePos + destPos) / 2.0;

QGraphicsTextItem\* weightItem = *new* QGraphicsTextItem('[' + QString::number(edges.count())

+ ']' + QString::number(weight));

weightItem->setPos(centerPos);

scene->addItem(*weightItem*);

edgeItems.append(lineItem);

weightItems.append(weightItem);

Edge edge{sourceNodeId, destNodeId, weight};

edges.append(edge);

++edgeCount;

}

}

void **deleteEdge**(int edgeId)

{

*if* (edgeId >= 0 && edgeId < edges.size())

{

scene->removeItem(*edgeItems[edgeId]*);

scene->removeItem(*weightItems[edgeId]*);

*delete* edgeItems[edgeId];

*delete* weightItems[edgeId];

edgeItems.remove(edgeId);

weightItems.remove(edgeId);

edges.remove(edgeId);

--edgeCount;

}

}

void **changeEdgeWeight**(int edgeId, int weight)

{

*if* (edgeId >= 0 && edgeId < edges.size())

{

weightItems[edgeId]->setPlainText('[' + QString::number(edges.count())

+ ']' + QString::number(weight));

edges[edgeId].weight = weight;

}

}

QVector<Edge> **getEdges**(){*return* edges;};

int **getNodeCount**(){*return* nodeCount;};

signals:

void **nodeCreated**(*const* QPointF& position);

void **nodeDeleted**(int nodeId);

void **edgeCreated**(int sourceNodeId, int destNodeId, int weight);

void **edgeDeleted**(int edgeId);

void **edgeWeightChanged**(int edgeId, int weight);

*protected*:

void ***contextMenuEvent***(QContextMenuEvent\* event) *override*

{

contextMenu->exec(event->globalPos());

}

*private* slots:

void **onCreateNodeActionTriggered**()

{

QPointF position = mapToScene(mapFromGlobal(QCursor::pos()));

emit nodeCreated(position);

}

void **onDeleteNodeActionTriggered**()

{

bool ok;

QString nodeIdStr = QInputDialog::getText(*this*, tr("Удалить вершину"), tr("Введите номер вершины:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !nodeIdStr.isEmpty())

{

int nodeId = nodeIdStr.toInt()-1;

emit nodeDeleted(nodeId);

}

}

void **onCreateEdgeActionTriggered**()

{

bool ok;

QString sourceIdStr = QInputDialog::getText(*this*, tr("Вставить ребро"), tr("Введите номер исходной вершины:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !sourceIdStr.isEmpty())

{

int sourceNodeId = sourceIdStr.toInt()-1;

QString destIdStr = QInputDialog::getText(*this*, tr("Вставить ребро"), tr("Введите номер конечной вершины:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !destIdStr.isEmpty())

{

int destNodeId = destIdStr.toInt()-1;

QString weightStr = QInputDialog::getText(*this*, tr("Вставить ребро"), tr("Введите вес ребра:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !weightStr.isEmpty())

{

int weight = weightStr.toInt();

emit edgeCreated(sourceNodeId, destNodeId, weight);

}

}

}

}

void **onDeleteEdgeActionTriggered**()

{

bool ok;

QString edgeIdStr = QInputDialog::getText(*this*, tr("Удалить ребро"), tr("Введите ID ребра:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !edgeIdStr.isEmpty())

{

int edgeId = edgeIdStr.toInt();

emit edgeDeleted(edgeId);

}

}

void **onChangeWeightActionTriggered**()

{

bool ok;

QString edgeIdStr = QInputDialog::getText(*this*, tr("Изменить вес"), tr("Введите ID ребра:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !edgeIdStr.isEmpty())

{

int edgeId = edgeIdStr.toInt();

QString weightStr = QInputDialog::getText(*this*, tr("Изменить вес"), tr("Введите новый вес:"), QLineEdit::*Normal*, "", *&ok*);

*if* (ok && !weightStr.isEmpty())

{

int weight = weightStr.toInt();

emit edgeWeightChanged(edgeId, weight);

}

}

}

*private*:

QGraphicsScene\* scene;

QMenu\* contextMenu;

QAction\* createNodeAction;

QAction\* deleteNodeAction;

QAction\* createEdgeAction;

QAction\* deleteEdgeAction;

QAction\* changeWeightAction;

QVector<QGraphicsEllipseItem\*> nodeItems;

QVector<QGraphicsTextItem\*> nodeIdItems;

QVector<QGraphicsLineItem\*> edgeItems;

QVector<QGraphicsTextItem\*> weightItems;

QVector<Edge> edges;

int nodeCount;

int edgeCount;

};

*class* **MainWindow** : *public* QWidget

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*) : QWidget(*parent*)

{

QVBoxLayout\* layout = *new* QVBoxLayout(*this*);

graphWidget = *new* Graph(*this*);

layout->addWidget(*graphWidget*);

QGridLayout\* buttonsLayout = *new* QGridLayout();

QPushButton\* bfsButton = *new* QPushButton("Обход в ширину");

QPushButton\* dfsButton = *new* QPushButton("Обход в глубину");

QPushButton\* deikstraButton = *new* QPushButton("Алгоритм Дейкстры");

buttonsLayout->addWidget(*bfsButton*, 0, 0);

buttonsLayout->addWidget(*dfsButton*, 0, 1);

buttonsLayout->addWidget(*deikstraButton*, 0, 2);

layout->addLayout(*buttonsLayout*);

connect(bfsButton, &QPushButton::clicked, *this*, &MainWindow::onBFSButtonClicked);

connect(dfsButton, &QPushButton::clicked, *this*, &MainWindow::onDFSButtonClicked);

connect(deikstraButton, &QPushButton::clicked, *this*, &MainWindow::onDeikstraButtonClicked);

}

*private* slots:

void **onBFSButtonClicked**()

{

int startNode = 0;

QVector<bool> visited(graphWidget->getNodeCount(), *false*);

std::queue<int> bfsQueue;

bfsQueue.push(startNode);

visited[startNode] = *true*;

*while* (!bfsQueue.empty())

{

int currentNode = bfsQueue.front();

bfsQueue.pop();

qDebug() << "[Обход в ширину] Посещена вершина:" << currentNode+1;

*for* (*const* Edge& edge : graphWidget->getEdges())

{

*if* (edge.source == currentNode && !visited[edge.destination])

{

bfsQueue.push(edge.destination);

visited[edge.destination] = *true*;

}

}

}

}

void **onDFSButtonClicked**()

{

int startNode = 0;

QVector<bool> visited(graphWidget->getNodeCount(), *false*);

std::stack<int> dfsStack;

dfsStack.push(startNode);

*while* (!dfsStack.empty()) {

int currentNode = dfsStack.top();

dfsStack.pop();

*if* (visited[currentNode]) {

*continue*;

}

visited[currentNode] = *true*;

qDebug() << "[Обход в глубину] Посещена вершина:" << currentNode+1;

*for* (*const* Edge& edge : graphWidget->getEdges())

{

*if* (edge.source == currentNode && !visited[edge.destination])

{

dfsStack.push(edge.destination);

}

}

}

}

void **onDeikstraButtonClicked**()

{

int startNode = 0;

QVector<int> distances(graphWidget->getNodeCount(), INT\_MAX);

distances[startNode] = 0;

QVector<bool> visited(graphWidget->getNodeCount(), *false*);

*for* (int i = 0; i < graphWidget->getNodeCount() - 1; ++i)

{

int minDistance = INT\_MAX;

int minNode = -1;

*for* (int j = 0; j < graphWidget->getNodeCount(); ++j)

{

*if* (!visited[j] && distances[j] < minDistance) {

minDistance = distances[j];

minNode = j;

}

}

*if* (minNode == -1) {

*break*;

}

visited[minNode] = *true*;

*for* (*const* Edge& edge : graphWidget->getEdges()) {

*if* (edge.source == minNode && distances[minNode] != INT\_MAX &&

distances[minNode] + edge.weight < distances[edge.destination]) {

distances[edge.destination] = distances[minNode] + edge.weight;

}

}

}

*for* (int i = 0; i < graphWidget->getNodeCount(); ++i) {

qDebug() << "Самое короткое расстояние от " << startNode+1 << "до" << i+1 << ":" << distances[i];

}

}

*private*:

Graph\* graphWidget;

};

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

QApplication app(*argc*, *argv*);

MainWindow mainWindow;

mainWindow.setGeometry(100, 100, 800, 600);

mainWindow.show();

*return* app.exec();

}

#include "main.moc"

Скриншот работы программы

