Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» (ИТАС)

Лабораторная работа

На тему

«Графы»

Выполнил

Студент группы ИВТ-23-1б

Адаев Даниил Дмитриевич

Проверил

Доцент кафедры ИТАС

Яруллин Д. В.

г. Пермь, 2024

Реализовать алгоритмы для собственного, придуманного самим автором, варианта двунаправленного графа, имеющего не менее 6 вершин.

Алгоритмы:
1. Обход в ширину.
2. Обход в глубину.
3. Алгоритм Дейкстры.
4. Выполнить задание своего варианта из методички:
Laby_Chast_3.docx (лаб работа по графам)

Требования:

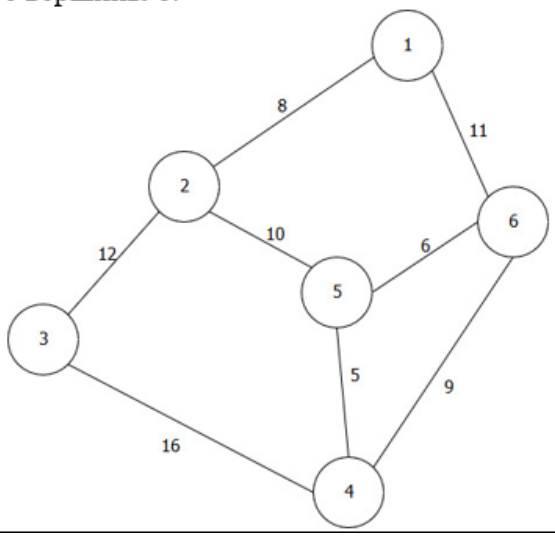
1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)

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).
- Реализовать вывод графа.

Вариант 2:

2	Реализовать граф, а также алгоритм Дейкстры, выполнив все необходимые действия. Выполнение начать с вершины 6. 

Код на c++

Graphs.pro

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
    graphs.cpp \
    main.cpp

HEADERS += \
    graphs.h

FORMS += \
    graphs.ui

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

graphs.h

```
#pragma once
#include <QMainWindow>
#include <QWidget>
#include <QPushButton>
#include <QMouseEvent>
#include <unordered_map>
#include <unordered_set>
#include "ui_graphs.h"

using namespace std;
```

```
struct Edge;

struct Node
{
    int data;
    vector<Edge*> edges;
    QPoint pos;
    Node() { pos = QPoint(350, 250); }
};

struct Edge
{
    int weight;
    Node* to;
    Node* from;
};

struct Graph
{
    vector<Node*> vnodes;

    unordered_map<int, Node*> nodes;

    void addNode(int data);
    void addEdge(int fromData, int toData, int weight);
    void clearGraph();
    void updateEdgeWeight(int startData, int endData, int newWeight);

    void removeNode(int data);
    void removeEdge(int startData, int endData);

    void DepthBypass(int startData, vector<int>& dbp);
    void WidthBypass(int startData, vector<int>& wbp);
    vector<int> AlgorithmBypass(int startData, int endData);
};

class Graphs : public QMainWindow
{
    Q_OBJECT

public:
    Graphs(QWidget *parent = nullptr);
    ~Graphs();
    Graph graph;

protected:
    void paintEvent(QPaintEvent* event) override;
    void mousePressEvent(QMouseEvent* event) override;
    void mouseMoveEvent(QMouseEvent* event) override;
    void mouseReleaseEvent(QMouseEvent* event) override;

private:
    Ui::Graphs *ui;
    Node* m_selectedNode;
    bool m_nodeSelected;

    bool sel = 0;
    Node* sNode;

    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void on_pushButton_3_clicked();
    void on_pushButton_4_clicked();
    void on_pushButton_5_clicked();
```

```
    void on_pushButton_6_clicked();
    void on_pushButton_7_clicked();
    void on_pushButton_8_clicked();
    void on_pushButton_9_clicked();
};
```

main.cpp

```
#include "graphs.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Graphs w;
    w.show();
    return a.exec();
}
```

graphs.cpp

```
#include "graphs.h"
#include "ui_graphs.h"
#include <cmath>
#include <QPainter>
#include <queue>
#include <stack>

void Graph::addNode(int data)
{
    if (nodes.find(data) == nodes.end())
    {
        Node* newNode = new Node;
        newNode->data = data;
        nodes[data] = newNode;
        vnodes.push_back(newNode);//
    }
}


void Graph::addEdge(int fromData, int toData, int weight)
{
    for (Edge* edge : nodes[fromData]->edges)
    {
        if (edge->to == nodes[toData])
        {
            return;
        }
    }

    Edge* newEdge = new Edge();
    newEdge->to = nodes[toData];
    newEdge->from = nodes[fromData];
    newEdge->weight = weight;
    nodes[fromData]->edges.push_back(newEdge);
}

void Graph::clearGraph()
{
    for (auto& pair : nodes)
    {
        Node* node = pair.second;
        delete node;
    }
    nodes.clear();
```

```
    vnodes.clear();
}

void Graph::updateEdgeWeight(int startData, int endData, int newWeight)
{
    if (nodes.find(startData) == nodes.end() || nodes.find(endData) ==
nodes.end())
    {
        return;
    }

    Node* startNode = nodes[startData];
    Node* endNode = nodes[endData];

    for (Edge* edge : startNode->edges)
    {
        if (edge->to == endNode)
        {
            edge->weight = newWeight;
            return;
        }
    }
}

void Graph::removeNode(int data)
{
    for (auto& pair : nodes)
    {
        Node* node = pair.second;
        vector<Edge*> edges_to_remove;

        for (Edge* edge : node->edges)
        {
            if (edge->to->data == data)
            {
                edges_to_remove.push_back(edge);
            }
        }

        for (Edge* edge : edges_to_remove)
        {
            auto it = find(node->edges.begin(), node->edges.end(), edge);
            if (it != node->edges.end())
            {
                node->edges.erase(it);
                delete edge;
            }
        }
    }

    auto it = nodes.find(data);
    if (it != nodes.end())
    {
        delete it->second;
        nodes.erase(it);
    }

    Node* nodeToRemove = nullptr;//
    for (Node* nodeTo : vnodes)
    {
        if (nodeTo->data == data)
        {
            nodeToRemove = nodeTo;
            break;
        }
    }
```

```
    if (nodeToRemove)
    {

        auto itn = find(vnodes.begin(), vnodes.end(), nodeToRemove);
        if (itn != vnodes.end())
        {
            vnodes.erase(itn);
        }
    }
}

void Graph::removeEdge(int startData, int endData)
{
    auto startNodeIt = nodes.find(startData);
    auto endNodeIt = nodes.find(endData);

    if (startNodeIt == nodes.end() || endNodeIt == nodes.end())
    {
        return;
    }

    Node* startNode = startNodeIt->second;
    Node* endNode = endNodeIt->second;

    Edge* edgeToRemove = nullptr;

    for (Edge* edge : startNode->edges)
    {
        if (edge->to->data == endData)
        {
            edgeToRemove = edge;
            break;
        }
    }

    if (edgeToRemove)
    {
        auto it = find(startNode->edges.begin(), startNode->edges.end(),
edgeToRemove);
        if (it != startNode->edges.end())
        {
            startNode->edges.erase(it);
            delete edgeToRemove;
        }
    }
}

Graphs::Graphs(QWidget *parent): QMainWindow(parent), ui(new Ui::Graphs)
{
    ui->setupUi(this);

    connect(ui->pushButton, &QPushButton::clicked, this,
&Graphs::on_pushButton_clicked);
    connect(ui->pushButton_2, &QPushButton::clicked, this,
&Graphs::on_pushButton_2_clicked);
    connect(ui->pushButton_3, &QPushButton::clicked, this,
&Graphs::on_pushButton_3_clicked);
    connect(ui->pushButton_4, &QPushButton::clicked, this,
&Graphs::on_pushButton_4_clicked);
    connect(ui->pushButton_5, &QPushButton::clicked, this,
&Graphs::on_pushButton_5_clicked);

    connect(ui->pushButton_6, &QPushButton::clicked, this,
&Graphs::on_pushButton_6_clicked);
```

```
    connect(ui->pushButton_7, &QPushButton::clicked, this,
&Graphs::on_pushButton_7_clicked);
    connect(ui->pushButton_8, &QPushButton::clicked, this,
&Graphs::on_pushButton_8_clicked);
    connect(ui->pushButton_9, &QPushButton::clicked, this,
&Graphs::on_pushButton_9_clicked);
}

Graphs::~Graphs()
{
    delete ui;
}

void Graphs::paintEvent(QPaintEvent* event)
{
    QPainter painter(this);
    QFont font = painter.font();
    font.setPointSize(16);
    painter.setFont(font);
    for (const auto& pair : graph.nodes)
    {
        Node* node = pair.second;
        for (Edge* edge : node->edges)
        {
            QPoint pos_f;
            QPoint pos_t;
            int d = 20 * sin(atan(1));
            double angles = atan2(-(edge->to->pos.y() - node->pos.y()),
(edge->to->pos.x() - node->pos.x()));
            pos_f = QPoint(node->pos.x() + 20 * cos(angles), node->pos.y() -
20 * sin(angles));
            pos_t = QPoint(edge->to->pos.x() - 20 * cos(angles), edge->to-
>pos.y() + 20 * sin(angles));
            painter.drawLine(pos_f, pos_t);
            int x_t = pos_f.x() + 4 * (pos_t.x() - pos_f.x()) / 5;
            int y_t = pos_f.y() - 4 * (pos_f.y() - pos_t.y()) / 5;
            painter.drawText(x_t - 10, y_t + 10, QString::number(edge-
>weight));

            QLine line(pos_f, pos_t);
            double angle = atan2(-line.dy(), line.dx()) - M_PI / 2;
            double arrowSize = 15;
            double arrowLength = 20;
            QPointF arrowP1 = pos_t + QPointF(sin(angle - M_PI / 10) *
arrowSize, cos(angle - M_PI / 10) * arrowSize);
            QPointF arrowP2 = pos_t + QPointF(sin(angle + M_PI / 10) *
arrowSize, cos(angle + M_PI / 10) * arrowSize);
            QPolygonF arrowHead;
            arrowHead << pos_t << arrowP1 << arrowP2;
            painter.drawPolygon(arrowHead);
        }
    }

    for (const auto& pair : graph.nodes)
    {
        Node* node = pair.second;
        painter.drawEllipse(node->pos, 20, 20);
        painter.drawText(node->pos.x() - 9, node->pos.y() + 8,
QString::number(node->data));
    }
    if (sel)
    {
        painter.drawEllipse(100, 100, 40, 40);
        painter.setBrush(Qt::green);
        painter.drawEllipse(sNode->pos, 20, 20);
```

```
        painter.drawText(sNode->pos.x() - 9, sNode->pos.y() + 8,
QString::number(sNode->data));
    }
}

void Graphs::mousePressEvent(QMouseEvent* event)
{
    if (event->button() == Qt::LeftButton)
    {
        m_nodeSelected = false;
        for (const auto& pair : graph.nodes)
        {
            Node* node = pair.second;
            if ((event->pos() - node->pos).manhattanLength() < 30)
            {
                m_selectedNode = node;
                m_nodeSelected = true;
                break;
            }
        }
        update();
    }
}

void Graphs::mouseMoveEvent(QMouseEvent* event)
{
    if (m_nodeSelected && m_selectedNode)
    {
        m_selectedNode->pos = event->pos();
        update();
    }
}
void Graphs::mouseReleaseEvent(QMouseEvent* event)
{
    if (event->button() == Qt::LeftButton && m_nodeSelected)
    {
        m_nodeSelected = false;
        m_selectedNode = nullptr;
        update();
    }
}

void Graph::DepthBypass(int startData, vector<int> &dbp)
{
    stack<Node*> nodeStack;
    nodeStack.push(nodes[startData]);//
    unordered_set<int> visited;
    visited.insert(startData);
    while (!nodeStack.empty())
    {
        Node* currentNode = nodeStack.top();
        nodeStack.pop();
        dbp.push_back(currentNode->data);
        for (Edge* edge : currentNode->edges)
        {
            if (visited.find(edge->to->data) == visited.end())
            {
                nodeStack.push(edge->to);
                visited.insert(edge->to->data);
            }
        }
    }
    for (auto const& pair : nodes)
    {
        if (visited.find(pair.first) == visited.end())
        {
```

```
            dbp.push_back(pair.first);
            visited.insert(pair.first);
        }
    }
}

void Graph::WidthBypass(int startData, vector<int> &wbp)
{
    queue<Node*> q;
    unordered_map<int, bool> visited;
    Node* startNode = nodes[startData];
    q.push(startNode);
    visited[startData] = true;
    while (!q.empty())
    {
        Node* currentNode = q.front();
        q.pop();
        wbp.push_back(currentNode->data);
        for (Edge* edge : currentNode->edges)
        {
            Node* neighborNode = edge->to;
            if (!visited[neighborNode->data])
            {
                visited[neighborNode->data] = true;
                q.push(neighborNode);
            }
        }
    }
    for (const auto& pair : nodes)
    {
        Node* node = pair.second;
        if (!visited[node->data])
        {
            q.push(node);
            visited[node->data] = true;
            while (!q.empty())
            {
                Node* currentNode = q.front();
                q.pop();
                wbp.push_back(currentNode->data);
                for (Edge* edge : currentNode->edges)
                {
                    Node* neighborNode = edge->to;
                    if (!visited[neighborNode->data])
                    {
                        visited[neighborNode->data] = true;
                        q.push(neighborNode);
                    }
                }
            }
        }
    }
}

vector<int> Graph::AlgorithmBypass(int startData, int endData)
{
    unordered_map<int, int> dist;
    unordered_map<int, int> prev;
    vector<int> result;
    for (auto& pair : nodes)
    {
        dist[pair.first] = INT_MAX;
        prev[pair.first] = -1;
    }
    dist[startData] = 0;
```

```
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> pq;
    pq.push({ 0, startData });
    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        if (u == endData) break;
        for (Edge* edge : nodes[u]->edges)
        {
            int v = edge->to->data;
            int alt = dist[u] + edge->weight;

            if (alt < dist[v])
            {
                dist[v] = alt;
                prev[v] = u;
                pq.push({ alt, v });
            }
        }
    }
    for (int at = endData; at != -1; at = prev[at])
    {
        result.push_back(at);
    }
    reverse(result.begin(), result.end());
    if (result[0] == endData) { result.pop_back(); }
    return result;
}

void Graphs::on_pushButton_clicked()
{
    QString text = ui->lineEdit->text();
    if (text.isEmpty()) return;

    int nodeValue = text.toInt();
    graph.addNode(nodeValue);
    ui->lineEdit->clear();
    update();
}

void Graphs::on_pushButton_2_clicked()
{
    if (ui->lineEdit_2->text().isEmpty() or ui->lineEdit_3->text().isEmpty()
or ui->lineEdit_4->text().isEmpty())
    {
        return;
    }
    int fromNode = ui->lineEdit_2->text().toInt();
    int toNode = ui->lineEdit_3->text().toInt();
    int weight = ui->lineEdit_4->text().toInt();
    if (graph.nodes.find(fromNode) != graph.nodes.end() &&
graph.nodes.find(toNode) != graph.nodes.end())
    {
        graph.addEdge(fromNode, toNode, weight);
        ui->lineEdit_2->clear();
        ui->lineEdit_4->clear();
        ui->lineEdit_3->clear();
        update();
    }
}
void Graphs::on_pushButton_3_clicked()
{
    if (ui->lineEdit_5->text().isEmpty())
    {
        return;
```

```
    }
    int del = ui->lineEdit_5->text().toInt();
    graph.removeNode(del);
    ui->lineEdit_5->clear();
    update();
}
void Graphs::on_pushButton_4_clicked()
{
    if (ui->lineEdit_6->text().isEmpty() or ui->lineEdit_7->text().isEmpty())
    {
        return;
    }
    int s = ui->lineEdit_6->text().toInt();
    int f = ui->lineEdit_7->text().toInt();
    graph.removeEdge(s, f);
    ui->lineEdit_7->clear();
    ui->lineEdit_6->clear();
    update();
}
void Graphs::on_pushButton_5_clicked()
{
    graph.addNode(1);
    graph.addNode(2);
    graph.addNode(3);
    graph.addNode(4);
    graph.addNode(5);
    graph.addNode(6);
    graph.addEdge(1, 2, 8);
    graph.addEdge(1, 6, 11);
    graph.addEdge(2, 3, 12);
    graph.addEdge(2, 5, 10);
    graph.addEdge(3, 4, 16);
    graph.addEdge(4, 5, 5);
    graph.addEdge(4, 6, 9);
    graph.addEdge(5, 6, 6);

    graph.addEdge(2, 1, 8);
    graph.addEdge(6, 1, 11);
    graph.addEdge(3, 2, 12);
    graph.addEdge(5, 2, 10);
    graph.addEdge(4, 3, 16);
    graph.addEdge(5, 4, 5);
    graph.addEdge(6, 4, 9);
    graph.addEdge(6, 5, 6);
    update();
}

void Graphs::on_pushButton_6_clicked()
{
    if (ui->lineEdit_8->text().isEmpty() or ui->lineEdit_9->text().isEmpty()
or ui->lineEdit_10->text().isEmpty())
    {
        return;
    }
    int s = ui->lineEdit_8->text().toInt();
    int t = ui->lineEdit_9->text().toInt();
    int w = ui->lineEdit_10->text().toInt();
    graph.updateEdgeWeight(s, t, w);
    ui->lineEdit_8->text().clear();
    ui->lineEdit_9->text().clear();
    ui->lineEdit_10->text().clear();
    update();
}

void Graphs::on_pushButton_7_clicked()
{
```

```
    graph.clearGraph();
    update();
}

void Graphs::on_pushButton_8_clicked()
{
    if( ui->lineEdit_11->text().isEmpty() ){ return; }

    int nodeStart = ui->lineEdit_11->text().toInt();
    vector<int> passedD;
    vector<int> passedW;
    QString bypassResult = "Обход в глубину:\n";

    graph.DepthBypass(nodeStart, passedD);

    for (unsigned int i = 0; i < passedD.size(); i++)
    {
        bypassResult.append(QString::number(passedD[i]));
        if (i < passedD.size() - 1)
        {
            bypassResult.append(", ");
        }
    }

    bypassResult += "\nОбход в ширину:\n";

    graph.WidthBypass(nodeStart, passedW);

    for (unsigned int i = 0; i < passedW.size(); i++)
    {
        bypassResult.append(QString::number(passedW[i]));
        if (i < passedW.size() - 1)
        {
            bypassResult.append(", ");
        }
    }

    ui->label_9->setText(bypassResult);

    ui->lineEdit_11->clear();
    ui->lineEdit_12->clear();
}

void Graphs::on_pushButton_9_clicked()
{
    if((ui->lineEdit_11->text().isEmpty())or(ui->lineEdit_12-
>text().isEmpty())){ return; }

    int nodeStart = ui->lineEdit_11->text().toInt();
    int nodeEnd = ui->lineEdit_12->text().toInt();
    vector<int> passed;
    QString bypassResult;

    passed = graph.AlgorithmBypass(nodeStart, nodeEnd);

    for (unsigned int i = 0; i < passed.size(); i++)
    {
        bypassResult.append(QString::number(passed[i]));
        if (i < passed.size() - 1)
        {
            bypassResult.append(", ");
        }
    }

    ui->label_9->setText("Алгоритм Дейкстры\n" + bypassResult);
    ui->lineEdit_11->clear();
```

```
    ui->lineEdit_12->clear();
}
```

graphs.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Graphs</class>
 <widget class="QMainWindow" name="Graphs">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1308</width>
    <height>659</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Graph</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QLabel" name="label_2">
    <property name="geometry">
     <rect>
      <x>990</x>
      <y>120</y>
      <width>55</width>
      <height>16</height>
     </rect>
    </property>
    <property name="text">
     <string>Куда</string>
    </property>
   </widget>
   <widget class="QLineEdit" name="lineEdit">
    <property name="geometry">
     <rect>
      <x>830</x>
      <y>10</y>
      <width>113</width>
      <height>31</height>
     </rect>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton_6">
    <property name="geometry">
     <rect>
      <x>1190</x>
      <y>140</y>
      <width>111</width>
      <height>31</height>
     </rect>
    </property>
    <property name="text">
     <string>Изменить ребро</string>
    </property>
   </widget>
   <widget class="QLineEdit" name="lineEdit_9">
    <property name="geometry">
     <rect>
      <x>950</x>
      <y>140</y>
      <width>113</width>
      <height>31</height>
     </rect>
    </property>
```

```
</widget>
<widget class="QLabel" name="label_3">
 <property name="geometry">
  <rect>
   <x>1100</x>
   <y>120</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Сколько</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_10">
 <property name="geometry">
  <rect>
   <x>1070</x>
   <y>140</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_7">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>280</y>
   <width>101</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>Очистить граф</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_3">
 <property name="geometry">
  <rect>
   <x>950</x>
   <y>90</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="label">
 <property name="geometry">
  <rect>
   <x>860</x>
   <y>120</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Откуда</string>
 </property>
</widget>
<widget class="QLabel" name="label_8">
 <property name="geometry">
  <rect>
   <x>1100</x>
   <y>170</y>
   <width>55</width>
```

```
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>Сколько</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_5">
  <property name="geometry">
   <rect>
    <x>990</x>
    <y>220</y>
    <width>55</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>Куда</string>
  </property>
 </widget>
 <widget class="QPushButton" name="pushButton_8">
  <property name="geometry">
   <rect>
    <x>950</x>
    <y>320</y>
    <width>131</width>
    <height>31</height>
   </rect>
  </property>
  <property name="text">
   <string>Обход</string>
  </property>
 </widget>
 <widget class="QLabel" name="label_6">
  <property name="geometry">
   <rect>
    <x>860</x>
    <y>170</y>
    <width>55</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>Откуда</string>
  </property>
 </widget>
 <widget class="QPushButton" name="pushButton_3">
  <property name="geometry">
   <rect>
    <x>950</x>
    <y>50</y>
    <width>101</width>
    <height>31</height>
   </rect>
  </property>
  <property name="text">
   <string>Удалить узел</string>
  </property>
 </widget>
 <widget class="QPushButton" name="pushButton_2">
  <property name="geometry">
   <rect>
    <x>1190</x>
    <y>90</y>
    <width>111</width>
    <height>31</height>
```

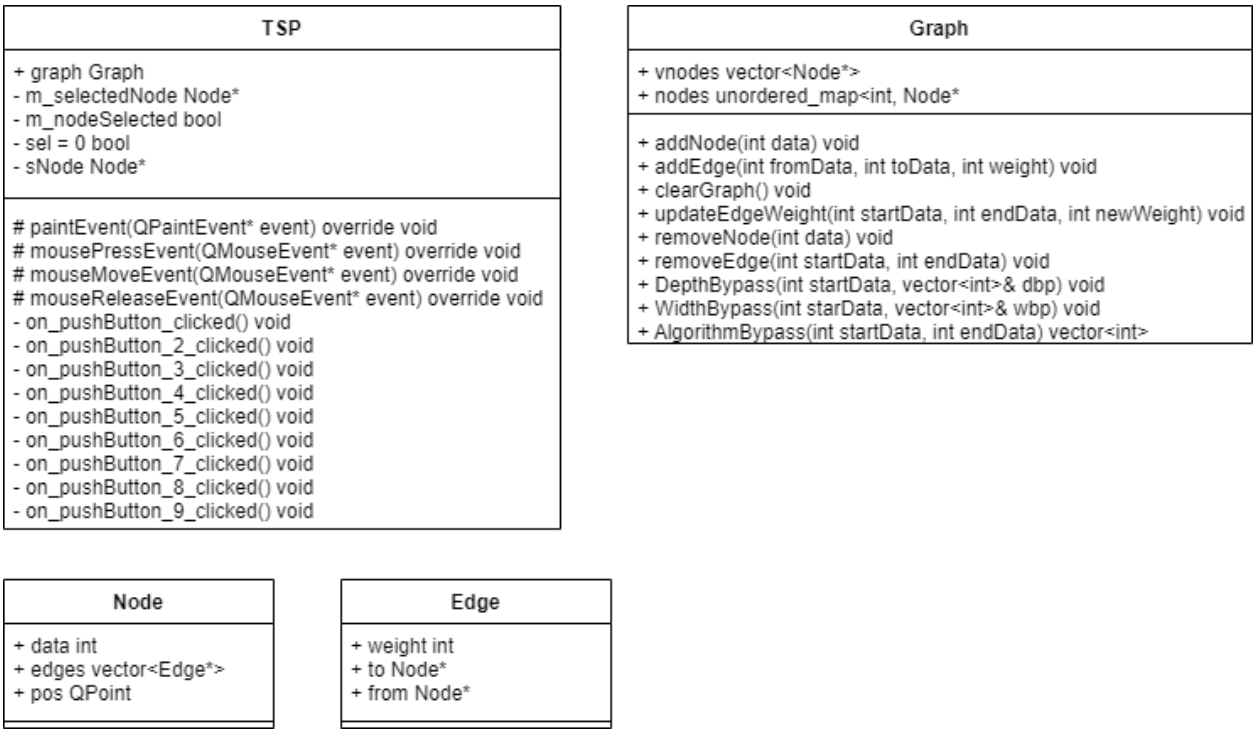
```
  </rect>
 </property>
 <property name="text">
  <string>Добавить ребро</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_5">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>50</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_5">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>240</y>
   <width>161</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>Создать граф варианта 2</string>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_4">
 <property name="geometry">
  <rect>
   <x>1070</x>
   <y>190</y>
   <width>101</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>Удалить ребро</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_8">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>140</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="pushButton">
 <property name="geometry">
  <rect>
   <x>950</x>
   <y>10</y>
   <width>101</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>Добавить узел</string>
 </property>
</widget>
<widget class="QLabel" name="label_7">
```

```
 <property name="geometry">
  <rect>
   <x>990</x>
   <y>170</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Куда</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_2">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>90</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_6">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>190</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="label_4">
 <property name="geometry">
  <rect>
   <x>860</x>
   <y>220</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Откуда</string>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_4">
 <property name="geometry">
  <rect>
   <x>1070</x>
   <y>90</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_7">
 <property name="geometry">
  <rect>
   <x>950</x>
   <y>190</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QTextBrowser" name="label_9">
```

```
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>420</y>
   <width>471</width>
   <height>211</height>
  </rect>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_11">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>320</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QLineEdit" name="lineEdit_12">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>370</y>
   <width>113</width>
   <height>31</height>
  </rect>
 </property>
</widget>
<widget class="QPushButton" name="pushButton_9">
 <property name="geometry">
  <rect>
   <x>950</x>
   <y>370</y>
   <width>131</width>
   <height>31</height>
  </rect>
 </property>
 <property name="text">
  <string>Алгоритм Дейкстры</string>
 </property>
</widget>
<widget class="QLabel" name="label_10">
 <property name="geometry">
  <rect>
   <x>860</x>
   <y>350</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Откуда</string>
 </property>
</widget>
<widget class="QLabel" name="label_11">
 <property name="geometry">
  <rect>
   <x>870</x>
   <y>400</y>
   <width>55</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string>Куда</string>
```

```
    </property>
   </widget>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

UML – диаграмма



Внешний вид программы: