Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе

Тема: «Графы»

Выполнил

Студент группы РИС-22-2б

Арапов М.В.

Проверил доц. Кафедры ИТАС

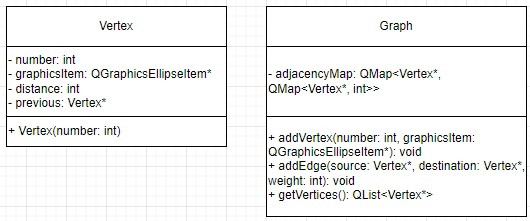
Полякова О. А.

Пермь 2023

**Постановка задачи**

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

**Uml-Диаграмма**



**Код программы**

#include <QApplication>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QGraphicsEllipseItem>

#include <QGraphicsLineItem>

#include <QGraphicsTextItem>

#include <QDebug>

#include <QQueue>

#include <QSet>

#include <QMap>

#include <cmath>

struct Vertex

{

int number;

QGraphicsEllipseItem\* graphicsItem;

int distance;

Vertex\* previous;

Vertex(int num)

: number(num), graphicsItem(nullptr), distance(INT\_MAX), previous(nullptr)

{

}

};

struct Graph

{

QMap<Vertex\*, QMap<Vertex\*, int>> adjacencyMap;

void addVertex(int number, QGraphicsEllipseItem\* graphicsItem)

{

Vertex\* vertex = new Vertex(number);

vertex->graphicsItem = graphicsItem;

adjacencyMap[vertex] = QMap<Vertex\*, int>();

}

void addEdge(Vertex\* source, Vertex\* destination, int weight)

{

adjacencyMap[source][destination] = weight;

adjacencyMap[destination][source] = weight;

}

QList<Vertex\*> getVertices() const

{

return adjacencyMap.keys();

}

};

void addLine(QGraphicsScene\* scene, const QPointF& start, const QPointF& end)

{

scene->addLine(QLineF(start, end), QPen(Qt::black));

}

class GraphView : public QGraphicsView

{

public:

GraphView(QWidget\* parent = nullptr)

: QGraphicsView(parent)

{

QGraphicsScene\* scene = new QGraphicsScene(this);

setScene(scene);

createGraph();

drawGraph();

runDijkstraAlgorithm(graph.getVertices().last());

findShortestPath(graph.getVertices().last(), graph.getVertices().first());

}

private:

void createGraph()

{

graph.addVertex(1, nullptr);

graph.addVertex(2, nullptr);

graph.addVertex(3, nullptr);

graph.addVertex(4, nullptr);

graph.addVertex(5, nullptr);

graph.addVertex(6, nullptr);

graph.addEdge(graph.getVertices().at(0), graph.getVertices().at(1), 8);

graph.addEdge(graph.getVertices().at(0), graph.getVertices().at(5), 11);

graph.addEdge(graph.getVertices().at(1), graph.getVertices().at(2), 12);

graph.addEdge(graph.getVertices().at(1), graph.getVertices().at(4), 10);

graph.addEdge(graph.getVertices().at(2), graph.getVertices().at(3), 16);

graph.addEdge(graph.getVertices().at(3), graph.getVertices().at(4), 5);

graph.addEdge(graph.getVertices().at(4), graph.getVertices().at(5), 6);

graph.addEdge(graph.getVertices().at(3), graph.getVertices().at(5), 9);

}

void drawGraph()

{

QGraphicsScene\* scene = this->scene();

QList<QPointF> vertexPositions = {

QPointF(50, 50),

QPointF(200, 50),

QPointF(350, 50),

QPointF(50, 200),

QPointF(200, 200),

QPointF(350, 200)

};

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

{

Vertex\* vertex = graph.getVertices().at(i);

QPointF position = vertexPositions.at(i);

QGraphicsEllipseItem\* vertexItem = new QGraphicsEllipseItem(-20, -20, 40, 40);

vertexItem->setPos(position);

scene->addItem(vertexItem);

vertex->graphicsItem = vertexItem;

QGraphicsTextItem\* numberTextItem = new QGraphicsTextItem(QString::number(vertex->number), vertexItem);

numberTextItem->setPos(-10, -10);

}

for (Vertex\* vertex : graph.getVertices())

{

for (auto it = graph.adjacencyMap[vertex].constBegin(); it != graph.adjacencyMap[vertex].constEnd(); ++it)

{

Vertex\* neighbor = it.key();

int weight = it.value();

QPointF start = vertex->graphicsItem->pos() + QPointF(20, 20);

QPointF end = neighbor->graphicsItem->pos() + QPointF(20, 20);

// Расстояние между вершинами

qreal distance = QLineF(start, end).length();

// Увеличение расстояния для размещения цифры веса

qreal weightDistance = 30;

// Расчет смещения цифры веса

qreal offsetX = (end.x() - start.x()) \* weightDistance / distance;

qreal offsetY = (end.y() - start.y()) \* weightDistance / distance;

// Расчет позиции цифры веса

QPointF weightPos = QPointF(start.x() + offsetX, start.y() + offsetY);

// Создание текстового элемента для цифры веса

QGraphicsTextItem\* weightTextItem = new QGraphicsTextItem(QString::number(weight));

weightTextItem->setPos(weightPos);

scene->addItem(weightTextItem);

// Рисование линии между вершинами

QGraphicsLineItem\* line = scene->addLine(QLineF(start, end));

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

}

}

}

void runDijkstraAlgorithm(Vertex\* startVertex)

{

qDebug() << "Running Dijkstra Algorithm from vertex" << startVertex->number;

// Initialize all vertices with infinite distance except the start vertex

for (Vertex\* vertex : graph.getVertices())

{

if (vertex == startVertex)

vertex->distance = 0;

else

vertex->distance = INT\_MAX;

}

QSet<Vertex\*> unvisitedVertices;

unvisitedVertices.unite(graph.getVertices().toSet());

while (!unvisitedVertices.isEmpty())

{

Vertex\* currentVertex = nullptr;

int minDistance = INT\_MAX;

// Find the unvisited vertex with the minimum distance

for (Vertex\* vertex : unvisitedVertices)

{

if (vertex->distance < minDistance)

{

currentVertex = vertex;

minDistance = vertex->distance;

}

}

if (currentVertex == nullptr)

break;

unvisitedVertices.remove(currentVertex);

// Update distances to neighbors

for (auto it = graph.adjacencyMap[currentVertex].constBegin(); it != graph.adjacencyMap[currentVertex].constEnd(); ++it)

{

Vertex\* neighbor = it.key();

int weight = it.value();

int totalDistance = currentVertex->distance + weight;

if (totalDistance < neighbor->distance)

{

neighbor->distance = totalDistance;

neighbor->previous = currentVertex;

}

}

}

}

void findShortestPath(Vertex\* startVertex, Vertex\* endVertex)

{

qDebug() << "Finding shortest path from vertex" << startVertex->number << "to vertex" << endVertex->number;

QList<Vertex\*> path;

Vertex\* currentVertex = endVertex;

while (currentVertex != nullptr)

{

path.prepend(currentVertex);

currentVertex = currentVertex->previous;

}

qDebug() << "Shortest path:";

for (Vertex\* vertex : path)

{

qDebug() << vertex->number;

}

}

Graph graph;

};

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

{

QApplication app(argc, argv);

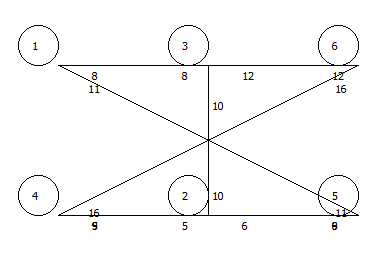
GraphView view;

view.show();

return app.exec();

}

**Вывод**

****