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

ПНИПУ

Лабораторная работа  
«Графы»

Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

**Введение**

Для выполнения лабораторной работы требуется сделать отчет программы. Создание программ – отличный способ практики программирования.

**Постановка задачи**

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

* Обход в ширину;
* Обход в глубину;
* Алгоритм Дейкстры.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности;
2. Визуализация графа с использованием любой доступной графической библиотеки;
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;

**Код программы**

**main.cpp**

#include <QApplication>

#include <QPushButton>

#include <QHBoxLayout>

#include "graphwidget.h"

#include <QInputDialog>

#include <QDebug>

#include <QPainter>

#include <QLabel>

// Функция для решения задачи коммивояжера методом полного перебора

int solveTravelingSalesmanProblem(const std::vector<std::vector<int>>& graph) {

int n = graph.size(); // Количество вершин графа

std::vector<int> vertexIndices(n);

for (int i = 0; i < n; ++i) {

vertexIndices[i] = i; // Индексы вершин

}

int minDistance = INT\_MAX; // Минимальное расстояние

do {

int currentDistance = 0; // Текущее расстояние

int currentVertex = vertexIndices[0]; // Текущая вершина

// Проходим по всем вершинам в порядке перестановки

for (int i = 1; i < n; ++i) {

int nextVertex = vertexIndices[i]; // Следующая вершина

currentDistance += graph[currentVertex][nextVertex]; // Добавляем расстояние между текущей и следующей вершиной

currentVertex = nextVertex; // Обновляем текущую вершину

}

// Добавляем расстояние от последней вершины к исходной

currentDistance += graph[currentVertex][vertexIndices[0]];

// Обновляем минимальное расстояние

minDistance = std::min(minDistance, currentDistance);

} while (std::next\_permutation(vertexIndices.begin() + 1, vertexIndices.end()));

return minDistance;

}

class GraphWidget2 : public QWidget {

public:

GraphWidget2(QWidget\* parent = nullptr) : QWidget(parent) {}

void paintEvent(QPaintEvent\* event) override {

Q\_UNUSED(event);

// Пример графа с весами ребер

std::vector<std::vector<int>> graph = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

int minDistance = solveTravelingSalesmanProblem(graph);

QPainter painter(this);

painter.setRenderHint(QPainter::Antialiasing);

// Рисуем вершины графа

int radius = 20;

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

int x = 50 + i \* 100;

int y = 50;

painter.drawEllipse(QPointF(x, y), radius, radius);

painter.drawText(QRectF(x - radius, y - radius, radius \* 2, radius \* 2), Qt::AlignCenter, QString::number(i));

}

// Рисуем ребра графа

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

for (int j = i + 1; j < int(graph.size()); ++j) {

int x1 = 50 + i \* 100;

int y1 = 50;

int x2 = 50 + j \* 100;

int y2 = 50;

painter.drawLine(QPointF(x1, y1), QPointF(x2, y2));

int weight = graph[i][j];

painter.drawText(QRectF((x1 + x2) / 2, (y1 + y2) / 2, 30, 30), Qt::AlignCenter, QString::number(weight));

}

}

// Выводим минимальное расстояние

painter.drawText(QRectF(50, 250, 300, 30), Qt::AlignLeft, "Min Distance: " + QString::number(minDistance));

}

};

using namespace std;

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

vector<std::vector<int>> graph = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

int minDistance = solveTravelingSalesmanProblem(graph);

qDebug() << "Minimal distance: " << minDistance << endl;

QApplication app(argc, argv);

//kommivoyajer

GraphWidget graphWidget;

GraphWidget2 graphWidget2;

QPushButton redrawButton("Redraw");

QObject::connect(&redrawButton, &QPushButton::clicked, [&graphWidget]() {

graphWidget.update();});

QHBoxLayout layout2;

layout2.addWidget(&redrawButton);

QWidget mainWidget2;

mainWidget2.setLayout(&layout2);

QVBoxLayout mainLayout2;

mainLayout2.addWidget(&graphWidget2);

mainLayout2.addWidget(&mainWidget2);

QWidget window2;

window2.setLayout(&mainLayout2);

window2.setWindowTitle("Traveling Salesman Problem");

window2.show();

// Create buttons for graph editing

QPushButton addButton("Add Vertex");

QPushButton removeButton("Remove Vertex");

QPushButton addEdgeButton("Add Edge");

QPushButton removeEdgeButton("Remove Edge");

QPushButton editMatrixButton("Edit Matrix");

QPushButton dfsButton("DFS");

QPushButton bfsButton("BFS");

QPushButton dijkstraButton("Dijkstra");

// Connect button signals to appropriate slots

QObject::connect(&addButton, &QPushButton::clicked, [&graphWidget]() {

// Prompt the user for vertex coordinates

int x = QInputDialog::getInt(nullptr, "Add Vertex", "Enter X coordinate:");

int y = QInputDialog::getInt(nullptr, "Add Vertex", "Enter Y coordinate:");

// Add the vertex to the graph

graphWidget.addVertex(x, y);

});

QObject::connect(&removeButton, &QPushButton::clicked, [&graphWidget]() {

// Prompt the user for the vertex index to remove

int index = QInputDialog::getInt(nullptr, "Remove Vertex", "Enter vertex index to remove:");

// Remove the vertex from the graph

graphWidget.removeVertex(index);

});

QObject::connect(&addEdgeButton, &QPushButton::clicked, [&graphWidget]() {

// Prompt the user for the source and destination vertices for the edge

int from = QInputDialog::getInt(nullptr, "Add Edge", "Enter source vertex:");

int to = QInputDialog::getInt(nullptr, "Add Edge", "Enter destination vertex:");

// Add the edge to the graph

graphWidget.addEdge(from, to);

});

QObject::connect(&removeEdgeButton, &QPushButton::clicked, [&graphWidget]() {

// Prompt the user for the source and destination vertices of the edge to remove

int from = QInputDialog::getInt(nullptr, "Remove Edge", "Enter source vertex:");

int to = QInputDialog::getInt(nullptr, "Remove Edge", "Enter destination vertex:");

// Remove the edge from the graph

graphWidget.removeEdge(from, to);

});

QObject::connect(&editMatrixButton, &QPushButton::clicked, [&graphWidget]() {

// Prompt the user for the new adjacency matrix

QVector<QVector<int>> matrix;

// Assuming a 6x6 matrix for this example

for (int i = 0; i < 6; ++i) {

QVector<int> row;

for (int j = 0; j < 6; ++j) {

int value = QInputDialog::getInt(nullptr, "Edit Matrix",

QString("Enter value for element (%1,%2):").arg(i).arg(j));

row.append(value);

}

matrix.append(row);

}

// Set the new adjacency matrix for the graph

graphWidget.setAdjacencyMatrix(matrix);

});

QObject::connect(&dfsButton, &QPushButton::clicked, [&graphWidget]() {

// Perform depth-first search on the graph

graphWidget.depthFirstSearch(0); // Start from vertex 0

});

QObject::connect(&bfsButton, &QPushButton::clicked, [&graphWidget]() {

// Perform breadth-first search on the graph

graphWidget.breadthFirstSearch(0); // Start from vertex 0

});

QObject::connect(&dijkstraButton, &QPushButton::clicked, [&graphWidget]() {

// Perform Dijkstra's algorithm on the graph

graphWidget.dijkstra(0, 5); // Start from vertex 0

});

// Create a layout for the buttons

QHBoxLayout layout;

layout.addWidget(&addButton);

layout.addWidget(&removeButton);

layout.addWidget(&addEdgeButton);

layout.addWidget(&removeEdgeButton);

layout.addWidget(&editMatrixButton);

layout.addWidget(&dfsButton);

layout.addWidget(&bfsButton);

layout.addWidget(&dijkstraButton);

// Create a main widget and set the layout

QWidget mainWidget;

mainWidget.setLayout(&layout);

// Create a main layout for the main widget

QVBoxLayout mainLayout;

mainLayout.addWidget(&graphWidget);

mainLayout.addWidget(&mainWidget);

// Set the main layout for the application window

QWidget window;

window.setLayout(&mainLayout);

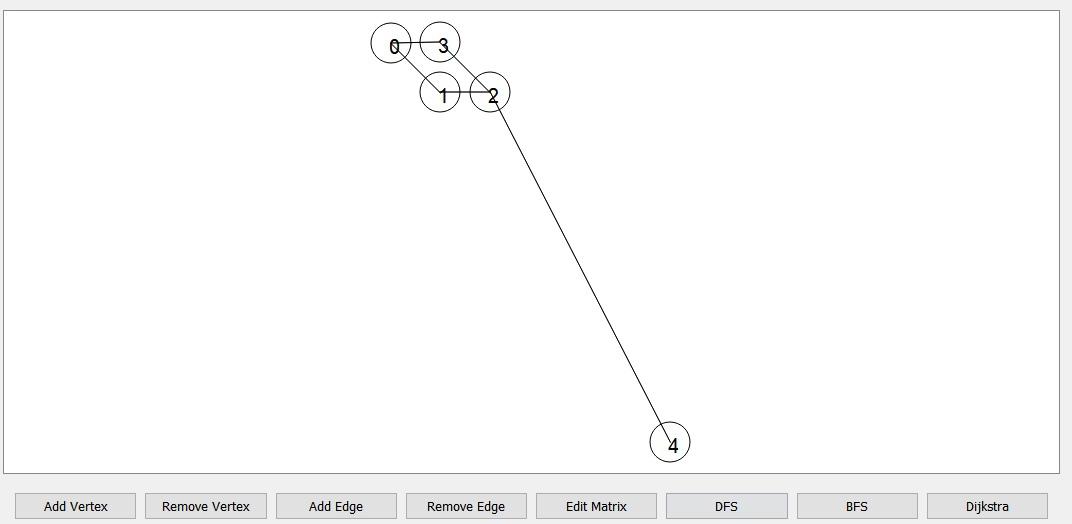
window.setWindowTitle("Graph Editor");

window.show();

return app.exec();

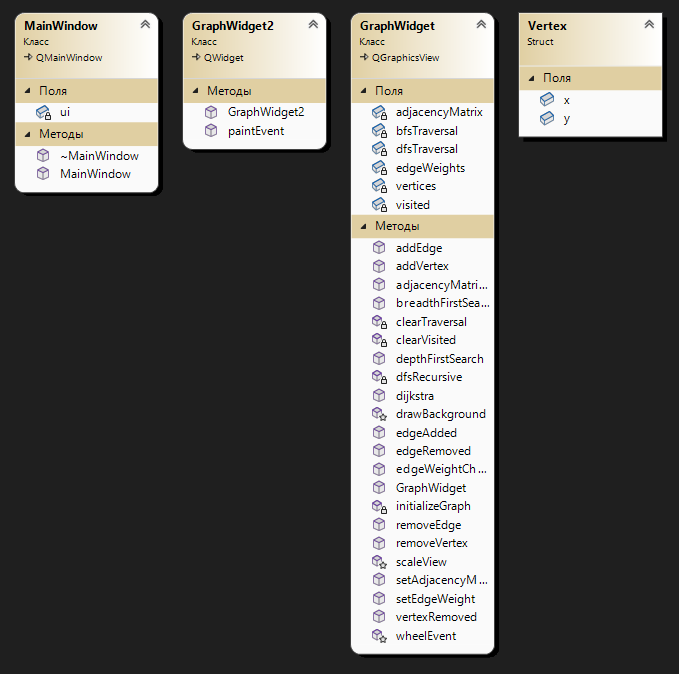
}

**Вывод программы**

****

***Рисунок 1 – вывод программы***

**UML диаграмма**

****

***Рисунок 2 – UML диаграмма***

**Вывод**

Программа выполняет свою задачу.

**Заключение**

Для решения задачи потребовались знания языка программирования, а конкретнее C++. Программа выполняет те условия, что были указаны в постановке задачи и работает без проблем.