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: «Задача Коммивояжера»

Выполнил

Студент группы РИС-24-1б

Конькова С. С.

Проверил доц. Кафедры ИТАС

Полякова О. А.

Пермь 2025

**Постановка задачи**

1) Создание и визуализация графа

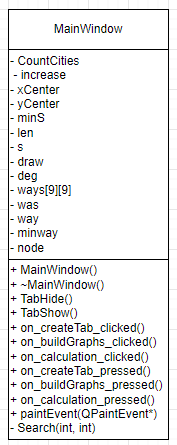
2) Выполнение задачи Коммивояжера

Принцип работы кода заключается в реализации алгоритма коммивояжера с использованием метода ветвей и границ.

Анализ

1. Использованы: Qt Framework (C++) для графического интерфейса, QGraphicsView/QGraphicsScene для визуализации, стандартная библиотека C++ (STL) для алгоритмов
2. Наследование от QGraphicsView для основного окна
3. Создание сцены (QGraphicsScene) для отображения элементов
4. Добавление кнопок управления (QPushButton)
5. Алгоритм решения TSP реализован методом полного перебора
6. Использует явно заданные веса рёбер при наличии
7. Автоматически вычисляет евклидово расстояние между точками при отсутствии явного ребра
8. Оптимальный путь выделяется синим цветом
9. Веса рёбер отображаются рядом с линиями
10. Точки нумеруются автоматически

**Uml-Диаграмма**



**Код программы**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsEllipseItem>

#include <QGraphicsLineItem>

#include <QGraphicsTextItem>

#include <QVector>

#include <algorithm>

#include <cmath>

#include <iostream>

#include <QPushButton>

#include <QInputDialog>

#include <QMessageBox>

#include <QTextEdit>

*struct* **Point** {

double x;

double y;

};

*struct* **Edge** {

int source;

int destination;

double weight;

};

*class* **GraphWidget** : *public* QGraphicsView {

*public*:

**GraphWidget**(QVector<Point>& points, QVector<Edge>& edges, QWidget\* parent = *nullptr*);

void **drawPoints**();

void **drawLines**();

void **solveTSP**();

void **drawOptimalPath**();

void **addPoint**();

void **removePoint**();

void **addEdge**();

void **removeEdge**();

*private*:

QGraphicsScene\* scene;

QVector<Point>& points;

QVector<Edge>& edges;

std::vector<int> optimalPath;

QTextEdit\* textOutput;

double minDistance;

void **updateMinDistance**() {

minDistance = std::numeric\_limits<double>::infinity();

solveTSP();

textOutput->clear();

textOutput->append("Minimum distance: " + QString::number(minDistance));

}

double **calculateDistance**(*const* Point& p1, *const* Point& p2) {

double dx = p1.x - p2.x;

double dy = p1.y - p2.y;

*return* std::sqrt(dx \* dx + dy \* dy);

}

};

#endif *//* *GRAPHWIDGET\_H*

#include "graphwidget.h"

#include <QApplication>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsEllipseItem>

#include <QGraphicsLineItem>

#include <QGraphicsTextItem>

#include <QVector>

#include <algorithm>

#include <cmath>

#include <iostream>

#include <QPushButton>

#include <QInputDialog>

#include <QMessageBox>

#include <QTextEdit>

GraphWidget::**GraphWidget**(QVector<Point>& points, QVector<Edge>& edges, QWidget\* parent)

: QGraphicsView(*parent*), scene(*new* QGraphicsScene(*this*)), points(points), edges(edges) {

scene->setSceneRect(-200, -200, 400, 400);

setScene(*scene*);

drawPoints();

drawLines();

setRenderHint(QPainter::*Antialiasing*);

setWindowTitle(tr("Traveling Salesman Problem"));

*//* *Создание* *поля* *для* *вывода* *текста*

textOutput = *new* QTextEdit(*this*);

textOutput->setReadOnly(*true*);

textOutput->setGeometry(10, 130, 150, 80);

solveTSP();

textOutput->append("Minimum distance: " + QString::number(minDistance));

QPushButton\* addPointButton = *new* QPushButton("Add Point", *this*);

addPointButton->move(10, 10);

connect(addPointButton, &QPushButton::clicked, *this*, &GraphWidget::addPoint);

QPushButton\* removePointButton = *new* QPushButton("Remove Point", *this*);

removePointButton->move(10, 40);

connect(removePointButton, &QPushButton::clicked, *this*, &GraphWidget::removePoint);

QPushButton\* addEdgeButton = *new* QPushButton("Add Edge", *this*);

addEdgeButton->move(10, 70);

connect(addEdgeButton, &QPushButton::clicked, *this*, &GraphWidget::addEdge);

QPushButton\* removeEdgeButton = *new* QPushButton("Remove Edge", *this*);

removeEdgeButton->move(10, 100);

connect(removeEdgeButton, &QPushButton::clicked, *this*, &GraphWidget::removeEdge);

}

void GraphWidget::**drawPoints**() {

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

*const* Point& p = points[i];

QGraphicsEllipseItem\* pointItem = scene->addEllipse(p.x - 3, p.y - 3, 6, 6);

pointItem->setBrush(Qt::*red*);

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

textItem->setPos(p.x + 5, p.y - 5);

}

}

void GraphWidget::**drawLines**() {

*for* (*const* Edge& edge : edges) {

*const* Point& p1 = points[edge.source];

*const* Point& p2 = points[edge.destination];

double distance = edge.weight;

*if* (distance > 0) {

QGraphicsLineItem\* lineItem = scene->addLine(p1.x, p1.y, p2.x, p2.y);

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

QGraphicsTextItem\* textItem = scene->addText(QString::number(distance));

textItem->setPos((p1.x + p2.x) / 2, (p1.y + p2.y) / 2);

}

}

}

void GraphWidget::**solveTSP**() {

int numPoints = points.size();

std::vector<int> path(numPoints);

*for* (int i = 0; i < numPoints; ++i) {

path[i] = i;

}

minDistance = std::numeric\_limits<double>::infinity();

*do* {

double distance = 0.0;

*for* (int i = 0; i < numPoints - 1; ++i) {

int source = path[i];

int destination = path[i + 1];

*auto* edgeIt = std::find\_if(edges.begin(), edges.end(), [&](*const* Edge& edge) {

*return* (edge.source == source && edge.destination == destination) ||

(edge.source == destination && edge.destination == source);

});

*if* (edgeIt != edges.end()) {

distance += edgeIt->weight;

} *else* {

distance += calculateDistance(points[source], points[destination]);

}

}

int lastSource = path[numPoints - 1];

int firstDestination = path[0];

*auto* edgeIt = std::find\_if(edges.begin(), edges.end(), [&](*const* Edge& edge) {

*return* (edge.source == lastSource && edge.destination == firstDestination) ||

(edge.source == firstDestination && edge.destination == lastSource);

});

*if* (edgeIt != edges.end()) {

distance += edgeIt->weight;

} *else* {

distance += calculateDistance(points[lastSource], points[firstDestination]);

}

*if* (distance < minDistance) {

minDistance = distance;

optimalPath = path;

}

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

std::cout << "Minimum distance: " << minDistance << std::endl;

drawOptimalPath();

}

void GraphWidget::**drawOptimalPath**() {

*if* (optimalPath.empty())

*return*;

QPen pen(Qt::*blue*, 2, Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*);

QGraphicsLineItem\* lineItem = *nullptr*;

*for* (int i = 0; i < optimalPath.size() - 1; ++i) {

int source = optimalPath[i];

int destination = optimalPath[i + 1];

*const* Point& p1 = points[source];

*const* Point& p2 = points[destination];

lineItem = scene->addLine(p1.x, p1.y, p2.x, p2.y, pen);

}

*//* *Connect* *the* *last* *and* *first* *points*

int lastSource = optimalPath.back();

int firstDestination = optimalPath.front();

*const* Point& p1 = points[lastSource];

*const* Point& p2 = points[firstDestination];

lineItem = scene->addLine(p1.x, p1.y, p2.x, p2.y, pen);

}

void GraphWidget::**addPoint**() {

bool ok;

double x = QInputDialog::getDouble(*this*, "Add Point", "Enter X coordinate:", 0, -1000, 1000, 1, *&ok*);

*if* (!ok)

*return*;

double y = QInputDialog::getDouble(*this*, "Add Point", "Enter Y coordinate:", 0, -1000, 1000, 1, *&ok*);

*if* (!ok)

*return*;

Point point;

point.x = x;

point.y = y;

points.push\_back(point);

QGraphicsEllipseItem\* pointItem = scene->addEllipse(x - 3, y - 3, 6, 6);

pointItem->setBrush(Qt::*red*);

QGraphicsTextItem\* textItem = scene->addText(QString::number(points.size()));

textItem->setPos(x + 5, y - 5);

solveTSP();

updateMinDistance();

}

void GraphWidget::**removePoint**() {

*if* (points.isEmpty()) {

QMessageBox::information(*this*, "Remove Point", "No points to remove.");

*return*;

}

bool ok;

int index = QInputDialog::getInt(*this*, "Remove Point", "Enter point index:", 1, 1, points.size(), 1, *&ok*);

*if* (!ok)

*return*;

index--; *//* *Adjust* *index* *to* *0-based*

points.remove(index);

scene->clear();

drawPoints();

drawLines();

solveTSP();

updateMinDistance();

}

void GraphWidget::**addEdge**() {

*if* (points.size() < 2) {

QMessageBox::information(*this*, "Add Edge", "At least two points are required to add an edge.");

*return*;

}

bool ok;

int sourceIndex = QInputDialog::getInt(*this*, "Add Edge", "Enter source point index:", 1, 1, points.size(), 1, *&ok*);

*if* (!ok)

*return*;

int destinationIndex = QInputDialog::getInt(*this*, "Add Edge", "Enter destination point index:", 1, 1, points.size(), 1, *&ok*);

*if* (!ok)

*return*;

double weight = QInputDialog::getDouble(*this*, "Add Edge", "Enter edge weight:", 0, 0, 1000, 1, *&ok*);

*if* (!ok)

*return*;

sourceIndex--; *//* *Adjust* *indices* *to* *0-based*

destinationIndex--;

edges.append({sourceIndex, destinationIndex, weight});

*const* Point& p1 = points[sourceIndex];

*const* Point& p2 = points[destinationIndex];

QGraphicsLineItem\* lineItem = scene->addLine(p1.x, p1.y, p2.x, p2.y);

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

QGraphicsTextItem\* textItem = scene->addText(QString::number(weight));

textItem->setPos((p1.x + p2.x) / 2, (p1.y + p2.y) / 2);

solveTSP();

updateMinDistance();

}

void GraphWidget::**removeEdge**() {

*if* (edges.isEmpty()) {

QMessageBox::information(*this*, "Remove Edge", "No edges to remove.");

*return*;

}

bool ok;

int sourceIndex = QInputDialog::getInt(*this*, "Remove Edge", "Enter source point index:", 1, 1, points.size(), 1, *&ok*);

*if* (!ok)

*return*;

int destinationIndex = QInputDialog::getInt(*this*, "Remove Edge", "Enter destination point index:", 1, 1, points.size(), 1, *&ok*);

*if* (!ok)

*return*;

sourceIndex--; *//* *Adjust* *indices* *to* *0-based*

destinationIndex--;

*auto* it = std::find\_if(edges.begin(), edges.end(), [&](*const* Edge& edge) {

*return* (edge.source == sourceIndex && edge.destination == destinationIndex) ||

(edge.source == destinationIndex && edge.destination == sourceIndex);

});

*if* (it != edges.end()) {

edges.erase(it);

scene->clear();

drawPoints();

drawLines();

solveTSP();

updateMinDistance();

} *else* {

QMessageBox::information(*this*, "Remove Edge", "Edge not found.");

}

}

#include <QApplication>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsEllipseItem>

#include <QGraphicsLineItem>

#include <QGraphicsTextItem>

#include <QVector>

#include <algorithm>

#include <cmath>

#include <iostream>

#include <QPushButton>

#include <QInputDialog>

#include <QMessageBox>

#include <QTextEdit>

#include "graphwidget.h"

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

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

QVector<Point> points;

points.append({100, 0});

points.append({0, 70});

points.append({50, 100});

points.append({150, 80});

points.append({100, 300});

points.append({-20, 200});

QVector<Edge> edges;

edges.append({0, 1, 17});

edges.append({0, 2, 21});

edges.append({0, 3, 48});

edges.append({1, 2, 25});

edges.append({2, 3, 6});

edges.append({3, 4, 13});

edges.append({4, 2, 8});

edges.append({4, 5, 40});

edges.append({5, 1, 3});

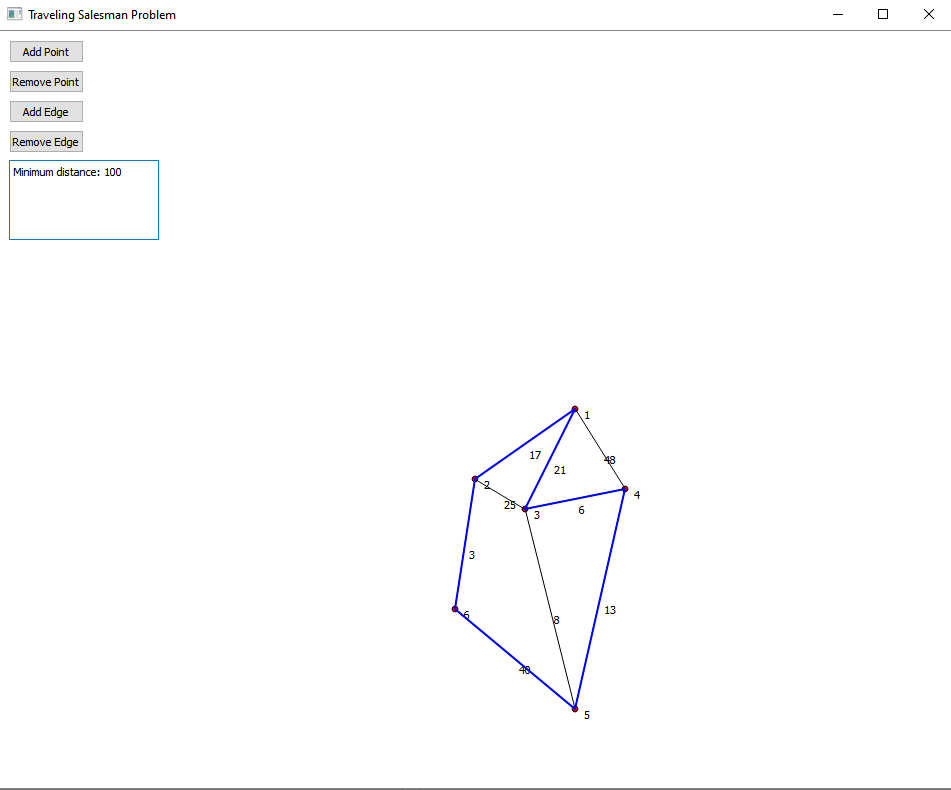
GraphWidget graphWidget(*points*, *edges*);

graphWidget.show();

*return* app.exec();

}

**Вывод**

****