МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Тихоокеанский государственный университет»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

«Изучение и реализация алгоритмов дискретной математики.»

«Решение задачи коммивояжера методом ветвей и границ.»

Текстовый документ курсовой работы

по дисциплине «Алгоритмы дискретной математики»

КР. 180009479.ТД

Выполнил студент Чекулаев В. Ю.

Факультет, группа ФКФН, ПО(аб) – 81

Руководитель работы Бахрушина Г.И.

Виза: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(доработать, к защите и т.д.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Хабаровск – 2020г.**Содержание**

[Введение 3](#_Toc43699321)

[Постановка задачи (в соответствии с вариантом) 4](#_Toc43699322)

[Основные понятия 5](#_Toc43699323)

[Описание алгоритма. 6](#_Toc43699324)

[Назначение программы 8](#_Toc43699325)

[Язык и среда программирования 8](#_Toc43699326)

[Описание основных структур данных, функций и методов 8](#_Toc43699327)

[Заключение 9](#_Toc43699328)

[Список используемой литературы 10](#_Toc43699329)

[Приложение А 11](#_Toc43699330)

[Приложение Б 29](#_Toc43699331)

# **Введение**

В жизни современных предприятий самого разного рода существенное место занимают транспортные потоки. Для осуществления своевременной доставки товара потребителям в кратчайшие сроки руководство компании занимается решением задачи кольцевого маршрута, иначе задачи коммивояжера.

Эта задача является упрощенной моделью для многих других задач дискретной оптимизации. В области оптимизации дискретных задач задача коммивояжёра служит катализатором, стимулирующим разработку наиболее эффективных методов, алгоритмов и способов их машинной реализации.

Целый ряд практических задач в области логистики сводится к классической задаче коммивояжера. Обилие эвристических методов её решения не означает отказа от возможности получения точных решений этой задачи. Классический алгоритм, реализующий метод ветвей и границ для решения задачи коммивояжёра, предложенный в 1963 году Дж. Литтлом, К. Мурти, Д. Суини и К. Кэролом и в настоящее время остаётся востребованным алгоритмом точного решения задачи нахождения гамильтонового цикла минимальной стоимости в полном взвешенном графе./3/

# **Основные понятия**

Граф, или ориентированный граф – это упорядоченная пара , где *V* – это непустое множество вершин или узлов, а *R* – множество (упорядоченных) пар различных вершин, называемых дугами или ориентированными ребрами.

Гамильтонов цикл в графе – это путь (цикл), который содержит все вершины графа. Граф называется графом Гамильтона в том случае, если он содержит цикл Гамильтона.

Гамильтонов путь – это простой путь (путь без петель), проходящий через каждую вершину графа ровно один раз. Гамильтонов путь отличается от цикла тем, что у пути начальные и конечные точки могут не совпадать, в отличие от цикла.

Взвешенным графом называют такой граф, в котором каждому ребру графа поставлено некоторое число , называемое весом ребра.

Матрица инцидентности – это одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро (дуга) и вершина). Столбцы матрицы соответствуют ребрам (дугам), а строки – вершинам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром (дугой). В данной работе матрица расстояний является матрицей инцидентности.

Согласно вышеприведенным определениям делается вывод, что решение проблемы коммивояжера эквивалентно поиску цикла Гамильтона с наименьшим весом в матрице.

# **Постановка задачи (в соответствии с вариантом)**

Задан полный ориентированный граф G = (V, E) с множеством вершин V = {1, …, n} и множеством дуг E. Каждой дуге (i,j) E приписана длина 0. В общем случае задача коммивояжера формулируется на произвольном графе, поэтому длины некоторых (в частности, не существующих) ребер могут быть сколь угодно большими. Так, считаем, что = .

Требуется найти гамильтонов контур минимальной длины.

# **Описание алгоритма.**

Одним из известных ранних алгоритмов точного решения ЗК для общего случая является алгоритм Литтла, основанный на методе ветвей и границ, который строит дерево решений для перебора вариантов маршрута (циклов обхода) с отсечением. Отсекаются такие частично построенные маршруты, у которых оценка снизу длины маршрута больше или равна длине ранее построенного полного наилучшего маршрута. При построении оценки снизу на каждом этапе работы алгоритма матрица расстояний подвергается такому преобразованию с трудоемкостью чтобы в каждой её строке и каждом столбце появился хотя бы один нуль.

Для вычисления оценки снизу длины частично построенного маршрута в алгоритме Литтла применяется следующее преобразование матрицы. Сначала из каждой строки вычитается её наименьший элемент, в результате в каждой строке образуется один (или больше) нулей. Затем из каждого столбца вычитается его наименьший элемент.

Это преобразование основано на том факте, что если из любого столбца или строки матрицы вычесть константу, то стоимость оптимального маршрута уменьшается на величину этой константы, а сам маршрут остается прежним. Сумма всех вычтенных при этом величин и будет оценкой снизу для всех вариантов маршрута, строящегося по данной матрице.

Следующее действие — выбор некоторого ребра графа, такого, что все возможные варианты маршрута разбиваются на две группы: те, которые включают выбранное ребро, и те, которые его не включают, и для каждой группы создается своя матрица расстояний. Эти матрицы подвергаются аналогичному преобразованию с выбором ребра и т. д. Таким образом, строящееся при этом дерево решений получается двоичным, каждому его узлу соответствует своя матрица расстояний.

Выбор ребра на очередном шаге производится таким образом, чтобы оптимальный вариант маршрута содержал выбранное ребро с наибольшей вероятностью. Для этого просматриваются строки матрицы и среди них выделяется та, в которой второе минимальное ребро имеет наибольший вес. Затем аналогичным образом просматриваются столбцы, и выделяется тот столбец, в котором второе минимальное ребро имеет наибольший вес. Окончательно выбирается такое ребро с нулевым весом, для которого вес второго минимального ребра в строке или столбце максимальный. Выбранное ребро включается в маршрут для вариантов маршрута 1-й группы и исключается из всех вариантов маршрута 2-й группы. В результате оценка снизу для всех вариантов маршрута 2-й группы увеличивается на вес второго минимального ребра в строке или столбце.

Рассмотрим более подробно реализацию алгоритма Литтла. На вход алгоритма подается квадратная матрица расстояний размером , все > 0, причём диагональные элементы = . Строки и столбцы матрицы помечаются отрезками маршрута, включёнными в строящийся вариант маршрута. В самом начале все эти отрезки суть отдельные вершины. Отрезок маршрута, начинающийся из вершины k и заканчивающийся в вершине i, задаётся парой чисел (k; i). После того как выбрано ребро (i; j), находящееся на пересечении строки, помеченной отрезком (k; i), и столбца, помеченного отрезком (j; l), для вариантов маршрута 1-й группы эти два отрезка объединяются. Для этого из матрицы расстояний удаляется строка (k; i) и столбец (j; l), в результате чего число строк и столбцов матрицы уменьшается на 1. Строка, помеченная отрезком, заканчивающимся в вершине l, и столбец, помеченный отрезком, начинающимся с вершины k, помечаются отрезком (k; l). Кроме того, элемент матрицы на их пересечении заменяется величиной . Это делается для того, чтобы предотвратить в дальнейшем выбор ребра графа (l; k), замыкающего локальный цикл, не включающий все вершины.

Для вариантов маршрута 2-й группы выбранное ребро (i; j) в матрице заменяется величиной , что предотвращает в последующем выбор этого ребра для маршрута. При этом размер матрицы не изменяется.

Когда размер матрицы уменьшится до , если при этом оценка снизу окажется меньше стоимости ранее найденного наилучшего маршрута, то строится окончательный замкнутый маршрут путем включения в него как ребра (i; j), так и ребра (l; k). Стоимость нового маршрута совпадает с его оценкой снизу, и он запоминается как наилучший маршрут среди всех ранее просмотренных.

Алгоритм Литтла просматривает строящееся таким образом дерево решений вглубь, в котором сначала делается переход к первой группе вариантов маршрута, а после их просмотра — ко второй группе.

Общая трудоёмкость алгоритма определяется произведением:

1. трудоёмкости преобразования матрицы и получения оценки снизу длины маршрута;
2. количеством просматриваемых узлов дерева решений.

Так как дерево решений двоичное, а его высота обычно ненамного превышает количество *n* вершин графа расстояний, то количество узлов в нём имеет порядок . Реальная же трудоёмкость алгоритма определяется эффективностью отсечений при просмотре дерева решений, её порядок — . /4-5/

# **Назначение программы**

Разработанная программа представляет собой визуальное приложение для нахождения гамильтонова цикла с минимальной стоимостью по заданной матрице инцидентности взвешенного ориентированного графа. Пользователь может задать таблицу расстояний непосредственно, либо разместить вершины графа на сцене (в этом случае за вес дуги будет принято расстояние между вершинами в пикселях). Код программы и примеры функционирования программы представлены в приложении А и Б соответственно.

# **Язык и среда программирования**

Программа была создана на языке C++ с использованием среды программирования (IDE) Qt Creator 4.12.2. /1-2/

# **Описание основных структур данных, функций и методов**

**class MainWindow** – класс главного окна;

**class Method –** класс модального окна для MainWindow;

**class MatrixD** – класс матрицы расстояний;

**class Node** – класс вершины графа;

**class Scene** – класс сцены для отрисовки графа;

**void Method::commivoyager(MatrixD)** – метод, в котором реализован алгоритм Литтла;

**int MatrixD::sum\_of\_const()** – метод вычисления оценки снизу;

**QPair<int, int> MatrixD::edge()** – метод выделения ребра разбиения;

**void MatrixD::reduction()** – метод редукции матрицы;

**void MatrixD::inf\_return\_trip()** – метод исключения обратного пути;

**void MatrixD::last\_transform()** – метод в котором выбираются последние два ребра для добавления в маршрут;

# **Заключение**

Выполнение курсовой работы завершилось созданием программного продукта, который обладает функциональными возможностями, определенными заданием. Для разработки программного продукта представлены два способа построения гамильтонова цикла с минимальной стоимостью. Первый способ подразумевает задание пользователем матрицы инцидентности, с помощью которой строится путь. Второй способ предполагает задание вершин графа пользователем с помощью представленных визуальных компонентов.

В приложении А представлен код программы. В приложении Б приведены скриншоты, полученные в результате тестирования программы.

# **Список используемой литературы**

1. doc.crossplatform.ru [Электронный ресурс]. Режим доступа: <http://doc.crossplatform.ru/qt/> (дата обращения 03.05.2020)
2. Р. Лафоре. Объектно-ориентированное программирование в С++, 4-Е издание. Издательство «Питер». 2004.
3. Копылова Е.С. , Николаева Д.С., Бунтова Е.В. Решение задачи коммивояжера с использованием метода ветвей и границ // Human progress. – 2018 . – Том 4, № 4 [электронный ресурс] URL: <http://progress-human.com/images/2018/Tom4_4/Kopylova.pdf>, свободный. – Загл. с экрана. – Яз. рус., агл.
4. Костюк Ю. Л. Эффективная реализация алгоритма решения задачи коммивояжера методом ветвей и границ. Национальный исследовательский Томский государственный университет, г. Томск, Россия. 2013.
5. Мудров В.И. Задача о коммивояжере. Издательство «Знание». Москва. 1969.

# **Приложение А**

Код программы

Содержимое файла main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Содержимое файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

void addLines(int);

private slots:

void on\_pushButton\_clicked();

void on\_checkBox\_2\_stateChanged(int arg1);

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Содержимое файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "method.h"

#include<QMessageBox>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_pushButton\_clicked()

{

int am\_of\_ci = ui->lineEdit->text().toInt();

if(am\_of\_ci < 3){

QMessageBox::critical(this, "Ошибка", "Введенное количество городов не может быть меньше 3-х");

return;

} else if(am\_of\_ci > 27){

QMessageBox::critical(this, "Ошибка", "Работа с большим числом городов не удобна в этом режиме");

return;

}

Method\* wind = new Method(this, am\_of\_ci, ui->checkBox\_2->isChecked());

wind->setModal(true);

wind->exec();

}

void MainWindow::on\_checkBox\_2\_stateChanged(int arg1)

{

if(arg1 == 0){

ui->lineEdit->setDisabled(false);

} else if(arg1 == 2){

ui->lineEdit->setDisabled(true);

}

}

Содержимое файла matrixd.h

#ifndef MATRIXD\_H

#define MATRIXD\_H

#include<QVector>

#include<QPair>

#include<QString>

class MatrixD

{

public:

MatrixD(QVector<QVector<QPair<int, QString>>>, QVector<int>, QVector<int>);

int sum\_of\_const();

void out() const;

void reduction(int, int);

void inf\_return\_trip();

void add\_inf(int, int);

QPair<int,int> edge();

size\_t get\_matrix\_size() const;

void add\_to\_path(QPair<int,int>);

void set\_bottomLimit(int);

int get\_bottomLimit() const;

int get\_str\_num(int) const;

int get\_col\_num(int) const;

QPair<int,QString> get\_data(int, int) const;

void set\_data(int, int, int);

void set\_data(int, int, QString);

QVector<QPair<int,int>> get\_path() const;

void last\_transform(int);

bool check\_path\_right() const;

bool all\_inf() const;

private:

QVector<QVector<QPair<int, QString>>> matrix;

QVector<int> table\_s;

QVector<int> table\_c;

QVector<QPair<int,int>> path;

int bottomLimit;

};

#endif // MATRIXD\_H

Содержимое файла matrixd.cpp

#include "matrixd.h"

#include<algorithm>

#include<iostream>

#include<iomanip>

#include<QString>

#include<QPair>

MatrixD::MatrixD(QVector<QVector<QPair<int, QString>>> m, QVector<int> t\_s, QVector<int> t\_c)

: matrix(m), table\_s(t\_s), table\_c(t\_c)

{

}

void MatrixD::reduction(int h, int k){

if(h < matrix.size() && k < matrix.size()){

if(!table\_s.isEmpty())

{

table\_s.remove(h);

}

if(!table\_c.isEmpty())

{

table\_c.remove(k);

}

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

for(int j = k; j < matrix.size()-1; j++){

matrix[i][j] = matrix[i][j+1];

}

}

for(int i = h; i < matrix.size()-1; i++){

for(int j = 0; j < matrix.size(); j++){

matrix[i][j] = matrix[i+1][j];

}

}

if(!matrix.isEmpty()){

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

matrix[i].remove(matrix.size() - 1);

}

matrix.remove(matrix.size() - 1);

}

}

}

QPair<int, int> MatrixD::edge(){

int min\_s = std::numeric\_limits<int>::max(), min\_c = std::numeric\_limits<int>::max(), max\_r = std::numeric\_limits<int>::min(), max\_r\_i;

QVector<QPair<int,int>> index;

QVector<int> rating;

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

for(int j = 0; j < matrix.size(); j++){

if(matrix[i][j].second != "inf" && matrix[i][j].first == 0){

for(int k = 0; k < matrix.size(); k++){

if(matrix[i][k].second != "inf" && k != j)

min\_s = std::min(min\_s, matrix[i][k].first);

}

if(min\_s == std::numeric\_limits<int>::max()){

min\_s = 0;

}

for(int k = 0; k < matrix.size(); k++){

if(matrix[k][j].second != "inf" && k != i)

min\_c = std::min(min\_c, matrix[k][j].first);

}

if(min\_c == std::numeric\_limits<int>::max()){

min\_c = 0;

}

rating.push\_back(min\_c+min\_s);

index.push\_back({i, j});

min\_s = std::numeric\_limits<int>::max();

min\_c = std::numeric\_limits<int>::max();

}

}

}

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

if(rating[i] > max\_r){

max\_r = rating[i];

max\_r\_i = i;

}

}

return index[max\_r\_i];

}

int MatrixD::sum\_of\_const(){

int subtractingSum = 0, min\_c = std::numeric\_limits<int>::max(), min\_s = std::numeric\_limits<int>::max();

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

for(int j = 0; j < matrix.size(); j++){

if(matrix[i][j].second != "inf"){

min\_s = std::min(min\_s, matrix[i][j].first);

}

}

if(min\_s == std::numeric\_limits<int>::max()){

min\_s = 0;

}

for(int j = 0; j < matrix.size(); j++){

if(matrix[i][j].second != "inf"){

matrix[i][j].first -= min\_s;

}

}

subtractingSum += min\_s;

min\_s = std::numeric\_limits<int>::max();

}

for(int j = 0; j < matrix.size(); j++){

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

if(matrix[i][j].second != "inf"){

min\_c = std::min(min\_c, matrix[i][j].first);

}

}

if(min\_c == std::numeric\_limits<int>::max()){

min\_c = 0;

}

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

if(matrix[i][j].second != "inf"){

matrix[i][j].first -= min\_c;

}

}

subtractingSum += min\_c;

min\_c = std::numeric\_limits<int>::max();

}

return subtractingSum;

}

size\_t MatrixD::get\_matrix\_size() const{

return matrix.size();

}

void MatrixD::inf\_return\_trip(){

if(!path.isEmpty()){

for(int i = 0; i < table\_s.size(); i++){

if(table\_s[i] == path.last().second) {

for(int j = 0; j < path.size(); j++){

for(int k = 0; k < table\_c.size(); k++){

if(path[j].first == table\_c[k]){

matrix[i][k].second = "inf";

}

}

}

}

}

}

}

void MatrixD::last\_transform(int amount\_of\_cities){

int start\_point{path[0].first};

bool path\_flag{false}, trash\_flag{false}, pop\_flag{true};

QVector<QPair<int,int>> temp, trash, buffer;

temp.push\_back(path[0]);

QPair<int,int> last(path[0]);;

while(temp.size() < amount\_of\_cities){

for(int j = 0; j < path.size(); j++){

if(temp.last().second == path[j].first){

for(int q = 0; q < buffer.size(); q++){

if(path[j] == buffer[q]){

goto l\_exit\_2;

}

}

temp.push\_back(path[j]);

if(temp.last().second == start\_point && temp.size() != amount\_of\_cities){

while(pop\_flag){

if(temp.last() == last){

pop\_flag = false;

} else{

temp.pop\_back();

}

}

while(!buffer.isEmpty()){

path.push\_back(buffer.last());

buffer.pop\_back();

}

pop\_flag = true;

temp.pop\_back();

trash.push\_back(last);

} else{

buffer.push\_back(temp.last());

path.remove(j);

path\_flag = true;

break;

}

}

l\_exit\_2:;

}

if(!path\_flag){

for(int k = 0; k < table\_s.size(); k++){

if(table\_s[k] == temp.last().second){

for(int l = 0; l < table\_c.size(); l++){

for(int m = 0; m < trash.size(); m++){

if(table\_s[k] == trash[m].first && table\_c[l] == trash[m].second){

trash\_flag = true;

}

}

if(!trash\_flag){

last = {table\_s[k], table\_c[l]};

temp.push\_back(last);

if(temp.last().first == temp.last().second){

trash.push\_back(last);

temp.pop\_back();

goto l\_exit\_1;

}

if(temp.size() == amount\_of\_cities && last.second != start\_point){

trash.push\_back(last);

temp.pop\_back();

goto l\_exit\_1;

}

for(int t = 0; t < temp.size()-1; t++){

if(temp.last().second == temp[t].first && temp.size() != amount\_of\_cities){

trash.push\_back(last);

temp.pop\_back();

goto l\_exit\_1;

}

}

goto l\_exit\_1;

}

trash\_flag = false;

}

}

}

}

l\_exit\_1:

path\_flag = false;

trash\_flag = false;

}

path = temp;

}

void MatrixD::add\_inf(int h, int k){

if(h < matrix.size() && k < matrix.size()){

matrix[h][k].second = "inf";

}

}

int MatrixD::get\_str\_num(int n) const {

if(n < table\_s.size()){

return table\_s[n];

} else{

std::cout << "MatrixD : get\_str\_num: index out of range\n";

exit(1);

}

}

int MatrixD::get\_col\_num(int n) const {

if(n < table\_c.size()){

return table\_c[n];

} else{

std::cout << "MatrixD : get\_col\_num: index out of range\n";

exit(1);

}

}

bool MatrixD::check\_path\_right() const{

bool flag{true};

if(!path.isEmpty()){

QVector<QPair<int,int>> temp;

for(int k = 0; k < path.size(); k++){

temp.push\_back(path[k]);

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

for(int j = 0; j < path.size(); j++){

if(temp.last().second == path[j].first){

temp.push\_back(path[j]);

if(temp.first().first == temp.last().second){

flag = false;

return flag;

}

}

}

}

temp.clear();

}

}

return flag;

}

bool MatrixD::all\_inf() const{

int count1 = 0, count2 = 0;

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

for(int j = 0; j < matrix.size(); j++){

if(matrix[i][j].second == "inf"){

count1++;

}

count2++;

}

}

if(count1 == count2){

return true;

} else{

return false;

}

}

void MatrixD::add\_to\_path(QPair<int, int> p){

path.push\_back(p);

}

void MatrixD::set\_bottomLimit(int b\_l){

bottomLimit = b\_l;

}

int MatrixD::get\_bottomLimit() const{

return bottomLimit;

}

QPair<int, QString> MatrixD::get\_data(int i, int j) const{

return matrix[i][j];

}

void MatrixD::set\_data(int i, int j, int x){

matrix[i][j].first = x;

}

void MatrixD::set\_data(int i, int j, QString x){

matrix[i][j].second = x;

}

QVector<QPair<int,int>> MatrixD::get\_path() const{

return path;

}

Содержимое файла method.h

#ifndef METHOD\_H

#define METHOD\_H

#include <QDialog>

#include<QVector>

#include<QLineEdit>

#include<QTextEdit>

#include"matrixd.h"

#include"qcustomplot.h"

#include"scene.h"

namespace Ui {

class Method;

}

class Method : public QDialog

{

Q\_OBJECT

public:

explicit Method(QWidget \*parent = nullptr,int am\_of\_ci = 2, bool chbx\_2 = false);

~Method();

void commivoyager(MatrixD);

private slots:

void backButtont\_clicked();

void calculateButton\_clicked();

void calculate\_button\_clicked();

void clearButtont\_clicked();

void againButtont\_clicked();

private:

Ui::Method \*ui;

QTextEdit\* textEdit;

QVector<QLineEdit\*> lines\_vector;

int amount\_of\_cities;

QLineEdit\* start\_point;

QCustomPlot\* graphic;

Scene\* scene;

QGraphicsView\* view;

};

#endif // METHOD\_H

Содержимое файла method.cpp

#include "method.h"

#include "ui\_method.h"

#include<QLineEdit>

#include<QGridLayout>

#include<QVBoxLayout>

#include<QHBoxLayout>

#include<QLabel>

#include<QString>

#include<QTextEdit>

#include<QPushButton>

#include<QVector>

#include<cmath>

#include<limits>

#include<QPair>

#include<algorithm>

#include<QLine>

#include<QMessageBox>

#include"matrixd.h"

#include"qcustomplot.h"

#include"scene.h"

#include"node.h"

#include<QGraphicsView>

#include<QGraphicsLineItem>

static QVector<QPair<int,int>> total\_path;

static int record = std::numeric\_limits<int>::max();

Method::Method(QWidget \*parent, int am\_of\_ci, bool chbx\_2) :

QDialog(parent),

amount\_of\_cities(am\_of\_ci),

ui(new Ui::Method)

{

ui->setupUi(this);

parentWidget()->hide();

if(!chbx\_2){

QVBoxLayout\* mainLay = new QVBoxLayout(this);

QHBoxLayout\* layButtons = new QHBoxLayout();

QGridLayout\* layLines = new QGridLayout();

QHBoxLayout\* layStart = new QHBoxLayout();

QTextEdit\* textEd = new QTextEdit("Введите расстояния в таблицу");

textEdit = textEd;

QLineEdit\* start = new QLineEdit;

start->setText("A");

start->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

start\_point = start;

QLabel\* s\_p = new QLabel("Начало маршрута: ");

s\_p->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

layStart->addWidget(s\_p);

layStart->addWidget(start);

QPushButton\* calculate = new QPushButton("Рассчитать", this);

connect(calculate, SIGNAL(clicked()), this, SLOT(calculateButton\_clicked()));

QPushButton\* back = new QPushButton("Назад", this);

back->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

connect(back, SIGNAL(clicked()), this, SLOT(backButtont\_clicked()));

layButtons->addWidget(calculate);

layButtons->addWidget(back);

int a\_o\_c = am\_of\_ci;

int letters\_c = 65, letters\_s = 65;

char c, s;

a\_o\_c++;

for(int i = 1; i < a\_o\_c\*a\_o\_c; i++)

{

if((i/a\_o\_c == 0) && (i%a\_o\_c == 0)){

c = ' ';

QLabel\* lbl = new QLabel(QString(c));

layLines->addWidget(lbl, i/a\_o\_c, i%a\_o\_c);

continue;

}

if(i/a\_o\_c == 0){

c = letters\_c;

QLabel\* lbl = new QLabel(QString(c));

lbl->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

layLines->addWidget(lbl, i/a\_o\_c, i%a\_o\_c);

letters\_c++;

continue;

}

if(i%a\_o\_c == 0){

s = letters\_s;

QLabel\* lbl = new QLabel(QString(s));

lbl->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);

layLines->addWidget(lbl, i/a\_o\_c, i%a\_o\_c);

letters\_s++;

continue;

}

QLineEdit\* led = new QLineEdit;

lines\_vector.push\_back(led);

layLines->addWidget(led, i/a\_o\_c, i%a\_o\_c);

}

for(int i = 0; i < a\_o\_c-1; i++){

lines\_vector[i\*a\_o\_c]->setText("inf");

lines\_vector[i\*a\_o\_c]->setEnabled(false);

}

mainLay->addWidget(textEd);

mainLay->addLayout(layStart);

mainLay->addLayout(layLines);

mainLay->addLayout(layButtons);

setLayout(mainLay);

} else if(chbx\_2){

QHBoxLayout\* mainVLayout = new QHBoxLayout(this);

QVBoxLayout\* lay = new QVBoxLayout();

QHBoxLayout\* lay2 = new QHBoxLayout();

QPushButton\* calculate\_button = new QPushButton("Рассчитать");

connect(calculate\_button, SIGNAL(clicked()), this, SLOT(calculate\_button\_clicked()));

QPushButton\* back\_button = new QPushButton("Назад");

connect(back\_button, SIGNAL(clicked()), this, SLOT(backButtont\_clicked()));

QPushButton\* clear\_button = new QPushButton("Очистить");

connect(clear\_button, SIGNAL(clicked()), this, SLOT(clearButtont\_clicked()));

QPushButton\* again\_button = new QPushButton("Задать вершины заново");

connect(again\_button, SIGNAL(clicked()), this, SLOT(againButtont\_clicked()));

QTextEdit\* txEd = new QTextEdit();

txEd->setMinimumWidth(300);

txEd->setMaximumWidth(300);

txEd->append("ПКМ - создать вершину");

txEd->append("ПКМ по вершине - удалить вершину");

txEd->append("ЛКМ по вершине + вести - переместить вершину");

textEdit = txEd;

Scene\* sc = new Scene(mainVLayout);

sc->setSceneRect(0,0, 490,490);

scene = sc;

QGraphicsView\* vi = new QGraphicsView(sc);

vi->setMinimumSize(500,500);

view = vi;

lay->addWidget(calculate\_button);

lay2->addWidget(clear\_button);

lay2->addWidget(again\_button);

lay->addLayout(lay2);

lay->addWidget(txEd);

lay->addWidget(back\_button);

mainVLayout->addWidget(vi);

mainVLayout->addLayout(lay);

setLayout(mainVLayout);

}

}

Method::~Method()

{

delete ui;

}

void Method::backButtont\_clicked(){

Node::id = 1;

parentWidget()->show();

this->close();

}

void Method::commivoyager(MatrixD matrix){

if(matrix.get\_matrix\_size() == 2){

matrix.last\_transform(amount\_of\_cities);

if(matrix.get\_bottomLimit() < record){

total\_path = matrix.get\_path();

record = matrix.get\_bottomLimit();

return;

} else{

return;

}

}

auto edge(matrix.edge());

auto matrix\_cpy1(matrix);

auto matrix\_cpy2(matrix);

matrix\_cpy1.add\_to\_path({matrix\_cpy1.get\_str\_num(edge.first), matrix\_cpy1.get\_col\_num(edge.second)});

if(!matrix\_cpy1.check\_path\_right()){

return;

}

matrix\_cpy1.reduction(edge.first, edge.second);

matrix\_cpy1.inf\_return\_trip();

if(matrix\_cpy1.all\_inf()){

return;

}

matrix\_cpy1.set\_bottomLimit(matrix.get\_bottomLimit() + matrix\_cpy1.sum\_of\_const());

if(matrix\_cpy1.get\_bottomLimit() > record){

return;

} else{

commivoyager(matrix\_cpy1);

}

matrix\_cpy2.add\_inf(edge.first, edge.second);

if(matrix\_cpy2.all\_inf()){

return;

}

matrix\_cpy2.set\_bottomLimit(matrix.get\_bottomLimit() + matrix\_cpy2.sum\_of\_const());

if(matrix\_cpy2.get\_bottomLimit() > record){

return;

} else{

commivoyager(matrix\_cpy2);

}

}

void Method::calculateButton\_clicked(){

QVector<QVector<QPair<int, QString>>> m(amount\_of\_cities, QVector<QPair<int,QString>>(amount\_of\_cities, {0, "0"}));

QVector<int> table(amount\_of\_cities);

for(int i = 0; i < amount\_of\_cities; i++){

table[i] = i;

}

int counter = 0;

for(int i = 0; i < amount\_of\_cities; i++){

for(int j = 0; j < amount\_of\_cities; j++){

if(lines\_vector[counter]->text() != "inf"){

m[i][j].first = lines\_vector[counter]->text().toInt();

counter++;

} else{

m[i][j].second = "inf";

counter++;

}

}

}

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

for(int j = 0; j < m.size(); j++){

if(m[i][j].first == 0 && i != j){

QMessageBox::warning(this, "Нулевое расстояние", "В одной из ячеек матрицы расстояний находится ноль,\nлибо одна из ячеек пуста.");

return;

}

}

}

MatrixD matrix(m, table, table);

matrix.set\_bottomLimit(matrix.sum\_of\_const());

commivoyager(matrix);

std::string s = start\_point->text().toStdString();

const char\* start = s.c\_str();

int c = 65, remember\_i = -1;

QVector<char> alphabet;

for(int i = 0; i < amount\_of\_cities; i++){

alphabet.push\_back(c);

if(c == std::toupper(\*start)) remember\_i = i;

c++;

}

if(remember\_i == -1){

QMessageBox::warning(this, "Начало пути", "Заданное начало пути не найдено.\nНачало пути принято за \"A\".");

remember\_i = 0;

}

QVector<char> result;

result.push\_back(alphabet[remember\_i]);

while(!total\_path.isEmpty()){

for(int i = 0; i < total\_path.size(); i++){

if(total\_path[i].first == remember\_i){

remember\_i = total\_path[i].second;

result.push\_back(alphabet[remember\_i]);

total\_path.remove(i);

}

}

}

textEdit->clear();

textEdit->append("Путь: ");

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

textEdit->insertPlainText(QString(result[i]) + "->");

}

textEdit->insertPlainText(QString(result[result.size()-1]));

textEdit->append("Вес пути: " + QString::number(record));

record = std::numeric\_limits<int>::max();

}

void Method::calculate\_button\_clicked(){

if(scene->getNodesAmount() < 3){

QMessageBox::critical(this, "Вершины", "Путь не может состоять из менее чем трех вершин");

return;

}

for(size\_t i = 0; i < scene->getNodesAmount(); i++){

for(size\_t j = 0; j < scene->getNodesAmount(); j++){

if(i != j)

if(scene->getNode(i)->collidesWithItem(scene->getNode(j))){

QMessageBox::warning(this, "Пересечения", "Имеются пересекающиеся вершины.");

return;

}

}

}

view->setEnabled(false);

amount\_of\_cities = scene->getNodesAmount();

QVector<QVector<QPair<int, QString>>> m(amount\_of\_cities, QVector<QPair<int,QString>>(amount\_of\_cities, {0, "0"}));

QVector<int> table(amount\_of\_cities);

for(int i = 0; i < amount\_of\_cities; i++){

table[i] = i;

}

for(int i = 0; i < amount\_of\_cities; i++){

for(int j = 0; j < amount\_of\_cities; j++){

if(i == j){

m[i][j].second = "inf";

} else{

m[i][j].first = std::sqrt(pow(scene->getNode(i)->x() - scene->getNode(j)->x(), 2) + pow(scene->getNode(i)->y() - scene->getNode(j)->y(), 2));

}

}

}

MatrixD matrix(m, table, table);

matrix.set\_bottomLimit(matrix.sum\_of\_const());

commivoyager(matrix);

textEdit->clear();

textEdit->insertPlainText("Путь:\n");

for(int i = 0; i < total\_path.size(); i++){

scene->addEdge(scene->getNode(total\_path[i].first)->x(), scene->getNode(total\_path[i].first)->y(),

scene->getNode(total\_path[i].second)->x(), scene->getNode(total\_path[i].second)->y());

if(i == total\_path.size()-1){

textEdit->insertPlainText(QString::number(scene->getNodeId(total\_path[i].first)) + "->" + QString::number(scene->getNodeId(total\_path[i].second)));

} else{

textEdit->insertPlainText(QString::number(scene->getNodeId(total\_path[i].first)) + "->");

}

}

textEdit->insertPlainText("\nДлина пути(в пикселях): " + QString::number(record));

}

void Method::clearButtont\_clicked(){

textEdit->clear();

textEdit->append("ПКМ - создать вершину");

textEdit->append("ПКМ по вершине - удалить вершину");

textEdit->append("ЛКМ по вершине + вести - переместить вершину");

scene->deleteEdges();

record = std::numeric\_limits<int>::max();

total\_path.clear();

view->setEnabled(true);

}

void Method::againButtont\_clicked(){

clearButtont\_clicked();

scene->deleteNodes();

Node::id = 1;

}

Содержимое файла node.h

#ifndef NODE\_H

#define NODE\_H

#include<QGraphicsEllipseItem>

class Node : public QGraphicsEllipseItem

{

public:

Node();

size\_t getSelfId() const;

void mouseMoveEvent(QGraphicsSceneMouseEvent \*event) override;

enum {Type = UserType + 1};

int type() const override {return Type;};

static size\_t id;

private:

size\_t self\_id;

int radius;

};

#endif // NODE\_H

Содержимое файла node.cpp

#include "node.h"

#include"scene.h"

#include<QGraphicsEllipseItem>

#include<QGraphicsSimpleTextItem>

#include<QGraphicsScene>

#include<QGraphicsSceneMouseEvent>

#include<QPen>

#include<QBrush>

int jump\_size{1};

Node::Node() : QGraphicsEllipseItem(-12,-12,12\*2,12\*2), radius(12)

{

setFlag(ItemIsMovable);

setFlag(ItemSendsGeometryChanges);

setCacheMode(DeviceCoordinateCache);

QBrush brush1(Qt::SolidPattern);

brush1.setColor(Qt::GlobalColor::white);

setBrush(brush1);

self\_id = id;

QGraphicsSimpleTextItem\* num = new QGraphicsSimpleTextItem(QString::number(id), this);

if(id < 10){

num->setPos(-3,-7);

} else if(id < 100){

num->setPos(-7,-7);

}

id++;

}

size\_t Node::id{1};

void Node::mouseMoveEvent(QGraphicsSceneMouseEvent \*event){

if(collidingItems().size() <= 1){

this->setPos(mapToScene(event->pos()));

} else{

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

if(collidingItems().toVector()[i]->type() == UserType + 1) {

if(collidingItems().toVector()[i]->pos().x() >= pos().x() && collidingItems().toVector()[i]->pos().y() >= pos().y()){

collidingItems().toVector()[i]->moveBy(jump\_size,jump\_size);

} else if(collidingItems().toVector()[i]->pos().x() >= pos().x() && collidingItems().toVector()[i]->pos().y() <= pos().y()){

collidingItems().toVector()[i]->moveBy(jump\_size,-jump\_size);

} else if(collidingItems().toVector()[i]->pos().x() <= pos().x() && collidingItems().toVector()[i]->pos().y() <= pos().y()){

collidingItems().toVector()[i]->moveBy(-jump\_size,-jump\_size);

} else if(collidingItems().toVector()[i]->pos().x() <= pos().x() && collidingItems().toVector()[i]->pos().y() >= pos().y()){

collidingItems().toVector()[i]->moveBy(-jump\_size,jump\_size);

}

}

}

}

}

size\_t Node::getSelfId() const{

return self\_id;

}

Содержимое файла scene.h

#ifndef SCENE\_H

#define SCENE\_H

#include<QGraphicsScene>

#include<QGraphicsItemGroup>

#include<QGraphicsLineItem>

#include"node.h"

class Scene : public QGraphicsScene{

Q\_OBJECT

public:

Scene(QObject \*parent = nullptr);

Node\* getNode(size\_t) const;

size\_t getNodeId(size\_t) const;

size\_t getNodesAmount() const;

void addEdge(qreal x1, qreal y1, qreal x2, qreal y2);

void deleteNodes();

void deleteEdges();

private:

void mousePressEvent(QGraphicsSceneMouseEvent \*event) override;

QVector<QGraphicsLineItem\*> edge\_v;

QVector<Node\*> node\_v;

};

#endif // SCENE\_H

Содержимое файла scene.cpp

#include "scene.h"

#include<QGraphicsSceneMouseEvent>

#include<QRectF>

#include"node.h"

Scene::Scene(QObject \*parent) : QGraphicsScene(parent)

{

}

void Scene::mousePressEvent(QGraphicsSceneMouseEvent \*event){

QGraphicsScene::mousePressEvent(event);

QPointF position = event->scenePos();

if(event->button() == Qt::RightButton){

Node\* gr = new Node();

gr->setPos(position);

for(Node\* elem : node\_v){

if(gr->collidesWithItem(elem)){

delete(gr);

//delete(elem);

Node::id--;

for(int i = 0; i < node\_v.size(); i++){

if(node\_v[i] == elem){

node\_v.remove(i);

delete(elem);

break;

}

}

return;

}

}

addItem(gr);

node\_v.push\_back(gr);

}

}

Node\* Scene::getNode(size\_t i) const {

return node\_v[i];

}

size\_t Scene::getNodeId(size\_t i) const{

return node\_v[i]->getSelfId();

}

size\_t Scene::getNodesAmount() const{

return node\_v.size();

}

void Scene::addEdge(qreal x1, qreal y1, qreal x2, qreal y2){

QGraphicsLineItem\* edge = new QGraphicsLineItem(x1, y1, x2, y2);

edge->setZValue(-1);

addItem(edge);

edge\_v.push\_back(edge);

}

void Scene::deleteNodes(){

for(int i = 0; i < node\_v.size(); i++){

delete(node\_v[i]);

}

node\_v.clear();

}

void Scene::deleteEdges(){

for(int i = 0; i < edge\_v.size(); i++){

delete(edge\_v[i]);

}

edge\_v.clear();

}

# **Приложение Б**

Функционирование программы

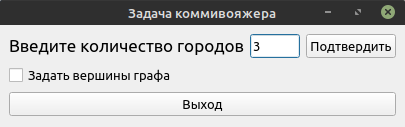


Рисунок Б.1 – Главное меню

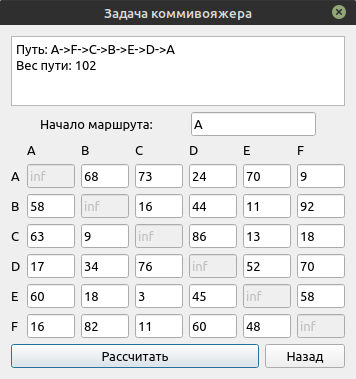


Рисунок Б.2 – Пример решения задачи для 6 городов

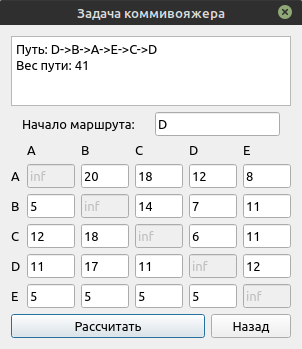


Рисунок Б.3 – Пример решения задачи для 5 городов

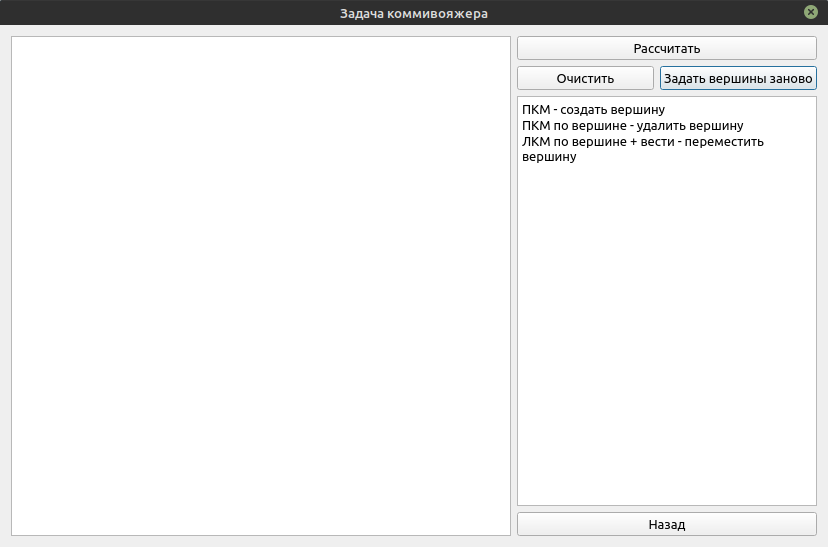


Рисунок Б.4 – Окно работы с вершинами графа

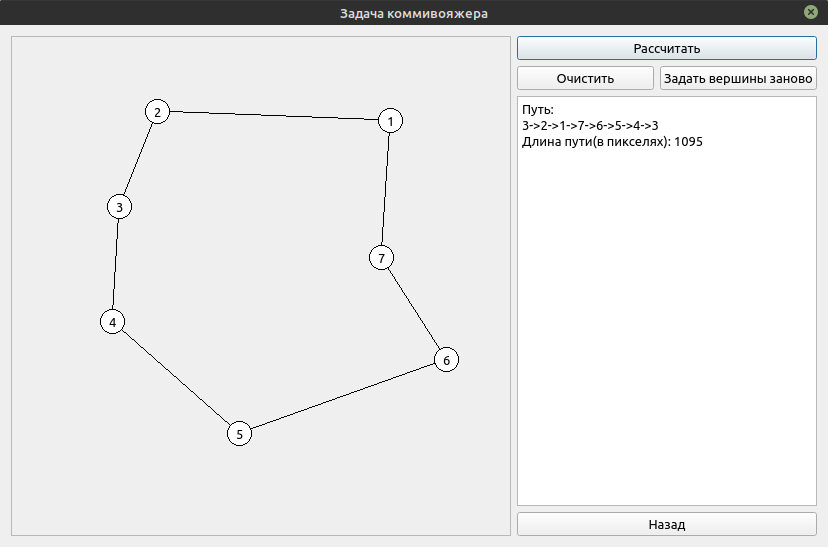


Рисунок Б.5 – Пример решения задачи путем задания 7 вершин графа

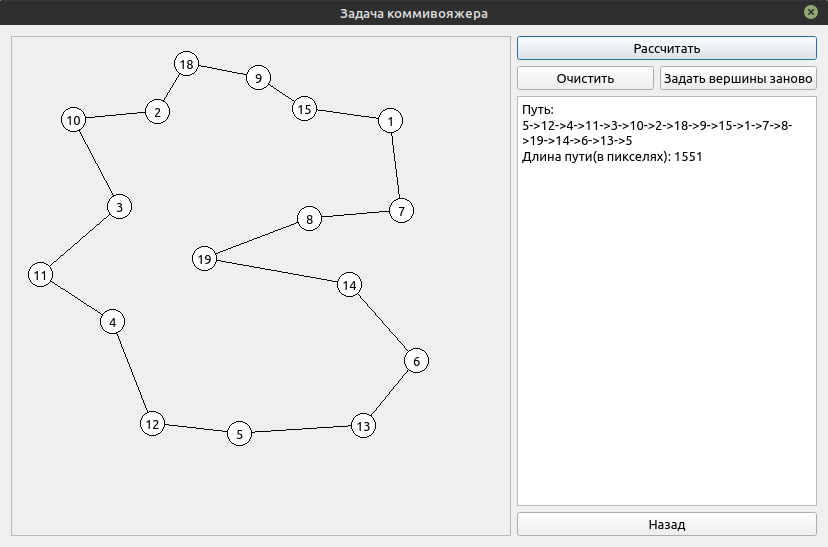


Рисунок Б.6 – Пример решения задачи путем задания 16 вершин графа