Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Кафедра «Информационные технологии и автоматизированные системы»

Творческая работа

«Решение задачи Коммивояжера»

Выполнил студент группы ИВТ-22-2б

Коняев Александр Сергеевич

Проверила доцент кафедры ИТАС

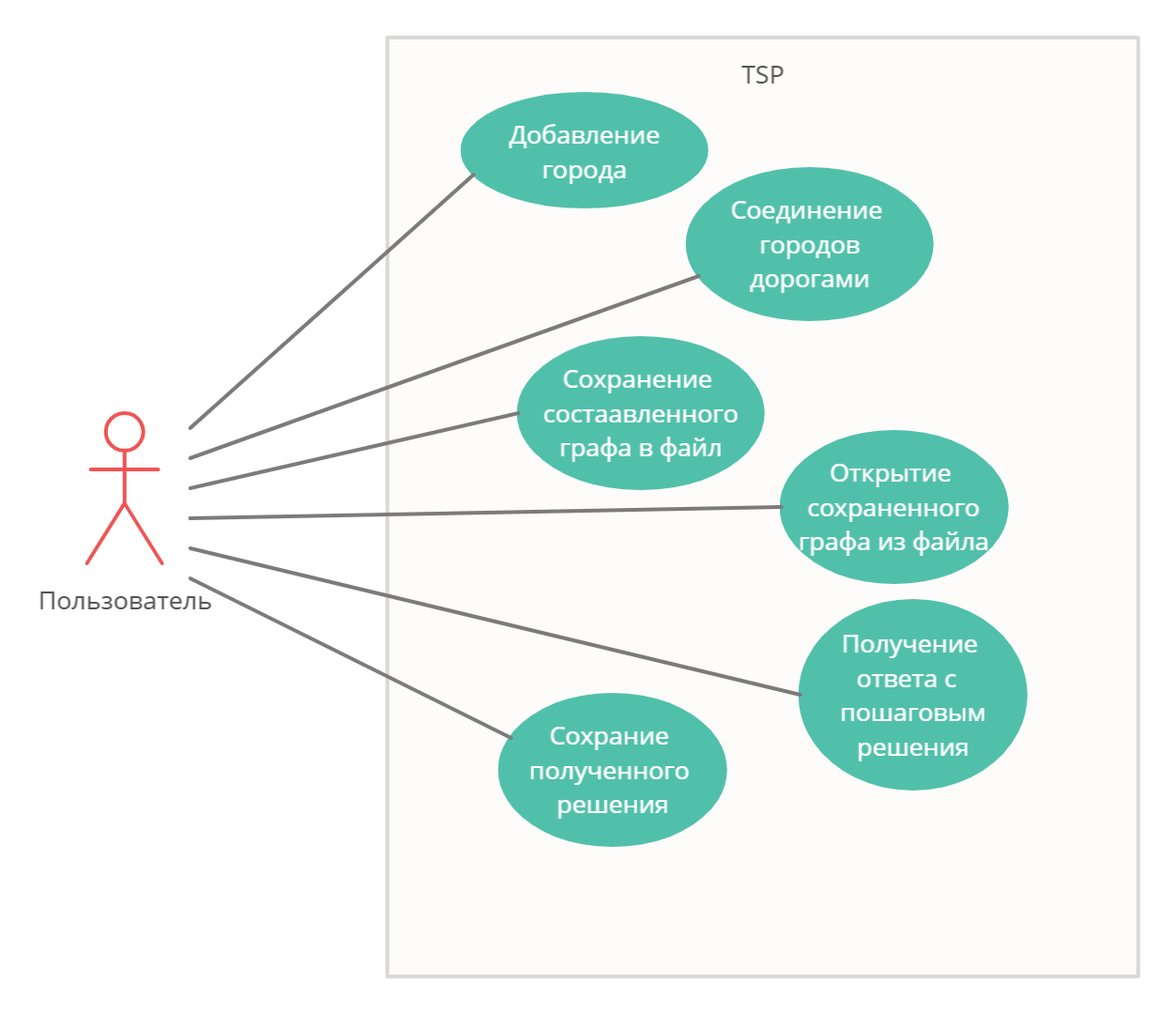
Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи:**

Задача заключается в поиске оптимального маршрута либо минимального времени, либо с наименьшим расстоянием, по которому движется Коммивояжер от пункта А с возвратом в этот же пункт. Причем, каждый город, встречающийся на пути Коммивояжера, должен быть посещен хотя бы один раз.

**Диаграмма вариантов:**



**Добавление дороги:** на экран должен добавляться новый круг-город с названием, заданным пользователем. Если название нового города совпадает с уже существующим, то он не добавится.

**Соединение городов дорогами:** пользователю должно быть предложено выбрать сначала первый, а затем второй город для соединения способом указанным пользователем (в обе или одну сторону) с указанным пользователем расстоянием-весом. Дорога не будет проложена, если первый и второй город совпадают, а также если дорога в данном направлении уже существует.

**Получение ответа с пошаговым решением:** грамма должна отобразить новое окно, в котором будет отображен ответ и пошаговое решение задачи с начальным городом указанным пользователем методом «Ветвей и границ». Если указанного города не существует, то следует вывести ошибку.

Код.

solution::solution(std::vector<city\*> cities, std::vector<road\*> roads,

QString starting\_city, QWidget \*parent) :

QDialog(parent),

ui(new Ui::solution)

{

ui->setupUi(this);

connect(ui->closeButton, &QPushButton::released, this, &solution::close);

int n = 0;

for (auto i : cities) {

city\_names.push\_back(i->name);

matrix.emplace\_back();

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

matrix[n].push\_back(-1);

}

for (int j = 0; j < n + 1; ++j) {

matrix[j].push\_back(-1);

}

++n;

}

solution\_text = "Для удобства присвоим городам номера:\n";

n = 1;

for (auto i : city\_names) {

solution\_text += i + ":" + QString::fromStdString(std::to\_string(n)) + ' ';

++n;

}

for (auto i : roads) {

int first\_city\_index = std::distance(city\_names.begin(),

std::find(city\_names.begin(),

city\_names.end(),

i->first\_city->name));

int second\_city\_index = std::distance(city\_names.begin(),

std::find(city\_names.begin(),

city\_names.end(),

i->second\_city->name));

matrix[first\_city\_index][second\_city\_index] = i->length;

}

int starting\_city\_index = std::distance(city\_names.begin(),

std::find(city\_names.begin(),

city\_names.end(),

starting\_city));

solution\_text += "\nМатрица смежности:\n";

print\_matrix(matrix, matrix.size());

route = solve(matrix, starting\_city\_index);

QString answer;

if (route.size() == city\_names.size()+1) {

for (auto i : route) {

answer += city\_names[i] + "->";

}

answer.resize(answer.size() - 2);

double length = 0;

int last\_city = route[0];

for (auto it = ++route.begin(); it != route.end(); it++) {

length += matrix[last\_city][\*it];

last\_city = \*it;

}

answer += ". Длина всего пути: " + double\_to\_str(length);

solution\_text += "\nМаршрут:" + answer;

} else {

solution\_text += "\nПосещены не все города! Искомого пути не существует!";

answer = "Нет такого пути!";

}

ui->answerLable->setText(ui->answerLable->text() + ' ' + answer);

ui->solutionBrowser->setText(solution\_text);

}

QString solution::double\_to\_str(double in){

QString result = QString::fromStdString(std::to\_string(in));

double intpart;

while (result.back() == '0'){

result.remove(result.length() - 1, 1);

}

if (modf(in, &intpart) == 0.0){

result.remove(result.length() - 1, 1);

}

return result;

}

std::vector<int> solution::solve(std::vector<std::vector<double>> matrix,

int starting\_city){

std::vector<std::pair<int, int>> branches;

std::vector<int> route;

matrix.emplace\_back();

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

matrix[i].emplace\_back();

matrix[matrix.size() - 1].emplace\_back();

}

matrix[matrix.size() - 1].push\_back(-1);

while (branches.size() < matrix.size() - 1) {

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

double min = INFINITY;

for (int j = 0; j < matrix.size() - 1; ++j) {

if (matrix[i][j] != -1 && matrix[i][j] < min) {

min = matrix[i][j];

}

}

matrix[i][matrix.size() - 1] = min;

}

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

for (int j = 0; j < matrix.size() - 1; ++j) {

if (matrix[i][j] != -1) {

matrix[i][j] -= matrix[i][matrix.size() - 1];

}

}

}

solution\_text += "\nМатрица после редукции столбцов:\n";

print\_matrix(matrix, matrix.size() - 1);

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

double min = INFINITY;

for (int j = 0; j < matrix.size() - 1; ++j) {

if (matrix[j][i] != -1 && matrix[j][i] < min) {

min = matrix[j][i];

}

}

matrix[matrix.size() - 1][i] = min;

}

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

for (int j = 0; j < matrix.size() - 1; ++j) {

if (matrix[j][i] != -1) {

matrix[j][i] -= matrix[matrix.size() - 1][i];

}

}

}

solution\_text += "\nМатрица после редукции строк:\n";

print\_matrix(matrix, matrix.size() - 1);

std::pair<int, int> branch;

double max\_sum = -1;

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

for (int j = 0; j < matrix.size() - 1; ++j) {

if (matrix[i][j] == 0) {

double min\_row = INFINITY, min\_column = INFINITY;

for (int k = 0; k < matrix.size() - 1; ++k) {

if (matrix[i][k] != -1 && k != j && matrix[i][k] < min\_row) {

min\_row = matrix[i][k];

}

if (matrix[k][j] != -1 && k != i && matrix[k][j] < min\_column) {

min\_column = matrix[k][j];

}

}

if (min\_row + min\_column > max\_sum) {

branch.first = i;

branch.second = j;

max\_sum = min\_row + min\_column;

}

}

}

}

solution\_text += "\nДобавим найденный путь: " + double\_to\_str(branch.first) +

"->" + double\_to\_str(branch.second) + ' ';

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

matrix[branch.first][i] = -1;

matrix[i][branch.second] = -1;

}

matrix[branch.first][branch.second] = -1;

matrix[branch.second][branch.first] = -1;

solution\_text += "\nМатрица после редукции матрицы:\n";

print\_matrix(matrix, matrix.size() - 1);

branches.push\_back(branch);

}

solution\_text += "\nПолученные пути:\n";

for (auto i : branches) {

solution\_text += double\_to\_str(i.first) + "->" + double\_to\_str(i.second) + ' ';

}

route.push\_back(starting\_city);

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

for (int j = 0; j < matrix.size() - 1; ++j) {

if (branches[j].first == route.back()) {

route.push\_back(branches[j].second);

break;

}

}

}

return route;

}

void solution::print\_matrix(std::vector<std::vector<double>> matrix, int size) {

std::vector<std::vector<QString>> matrix\_to\_print;

std::vector<std::vector<QString>> matrix\_separetion;

std::vector<int> max\_size;

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

std::vector<QString> \*new\_vec = new std::vector<QString>;

matrix\_to\_print.push\_back(\*new\_vec);

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

if (matrix[i][j] == -1) {

matrix\_to\_print[i].push\_back("\*");

} else {

matrix\_to\_print[i].push\_back(double\_to\_str(matrix[i][j]));

}

}

}

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

matrix\_to\_print[i].insert(matrix\_to\_print[i].begin(), double\_to\_str(i + 1));

}

matrix\_to\_print.emplace(matrix\_to\_print.begin());

matrix\_to\_print[0].push\_back("");

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

matrix\_to\_print[0].push\_back(double\_to\_str(i + 1));

}

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

max\_size.push\_back(0);

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

if (matrix\_to\_print[j][i].length() > max\_size[i]) {

max\_size[i] = matrix\_to\_print[j][i].length();

}

}

}

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

std::vector<QString> \*new\_vec = new std::vector<QString>;

matrix\_separetion.push\_back(\*new\_vec);

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

matrix\_separetion[i].push\_back("");

for (int k = 0; k < max\_size[j] - matrix\_to\_print[i][j].length(); ++k){

matrix\_separetion[i][j] += ' ';

}

}

}

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

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

solution\_text += matrix\_to\_print[i][j] + ' ' + matrix\_separetion[i][j];

}

solution\_text += '\n';

}

}

void solution::close() {

solution::~solution();

}

void solution::return\_solution() {

if (route.size() == city\_names.size()+1) emit solved(route);

}

Дизайн

При выполнении творческой работы мною был выбран минималистический дизайн с холодными тонами.

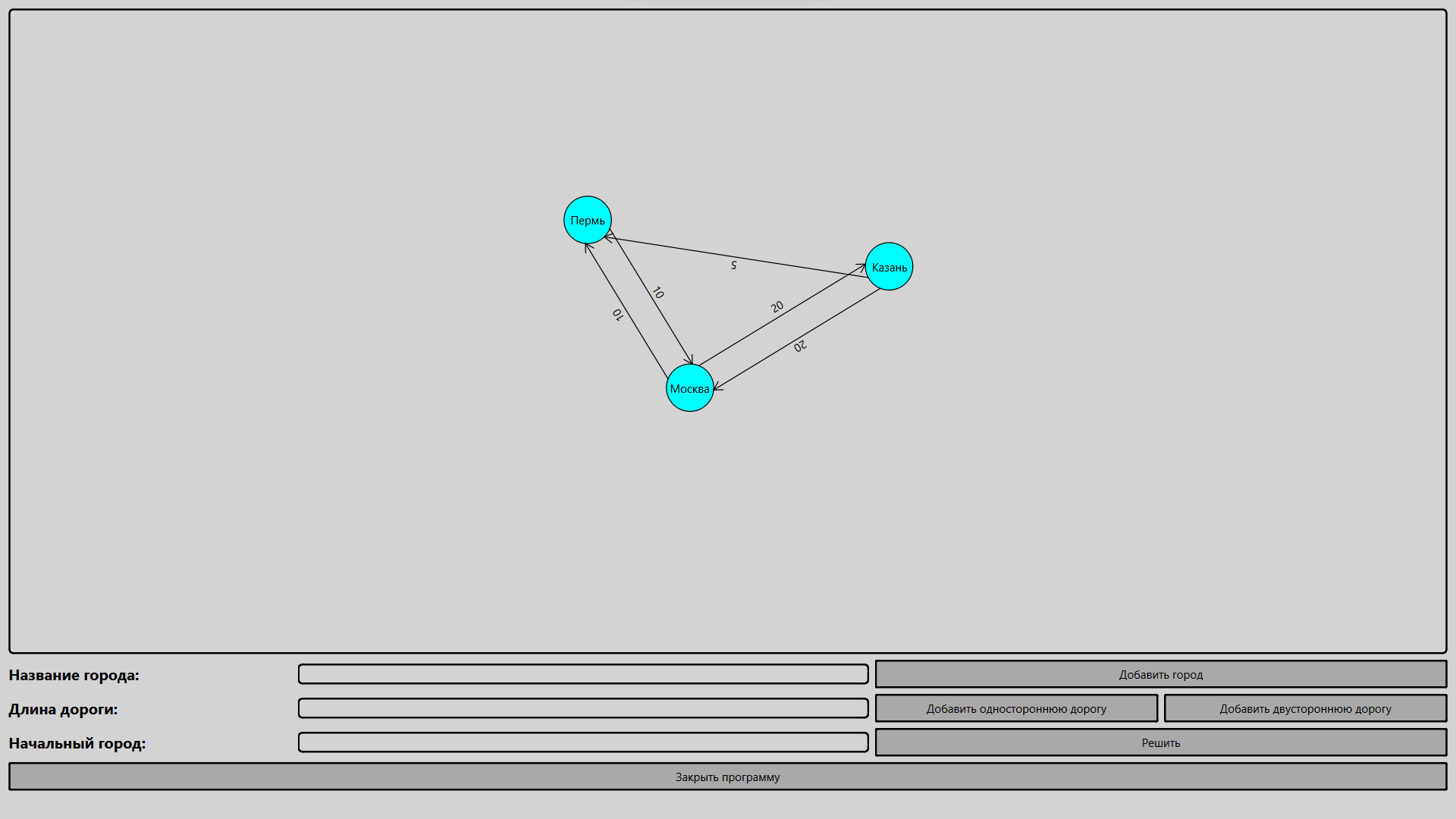


Рисунок 1.

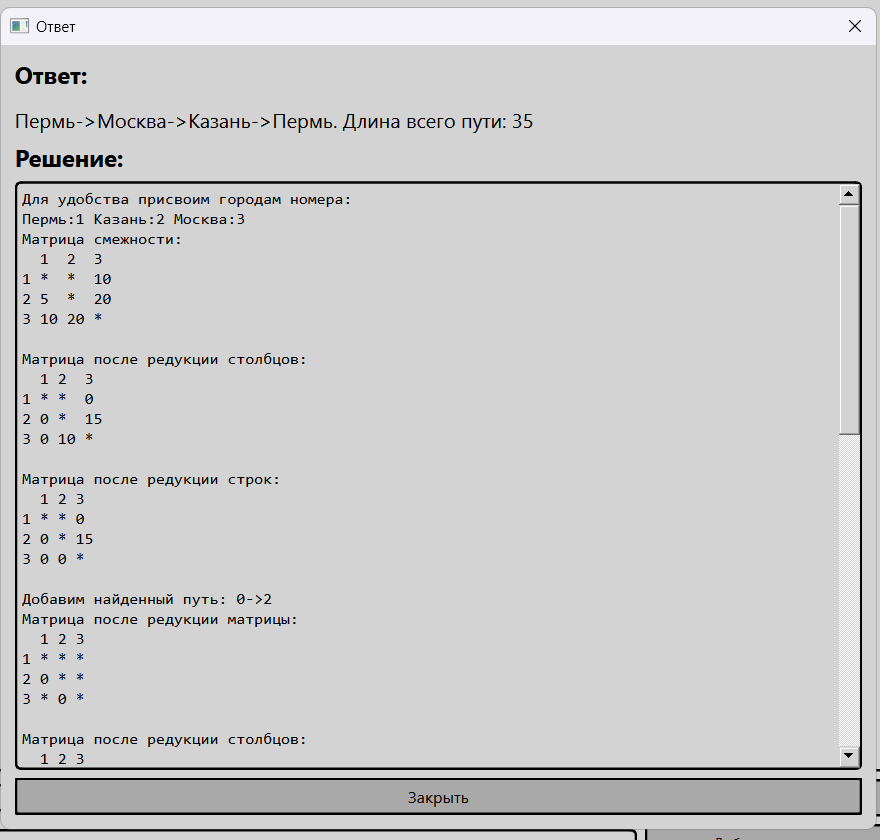


Рисунок 2.