ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа “Задача коммивояжёра”.

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

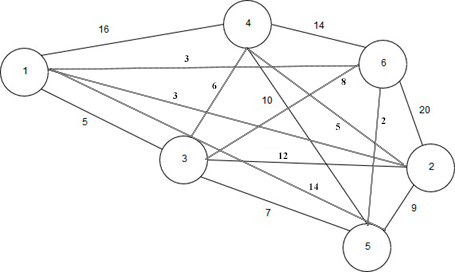
Проверила доцент кафедры ИТАС О.А. Полякова

2024

1. Постановка задачи

Имеется 6 городов. Выезжая из исходного города, коммивояжер должен побывать во всех городах по одному разу и вернуться в начальный город. Задача заключается в определении последовательности объезда городов, при которой коммивояжеру требуется минимизировать некоторый критерий эффективности: стоимость проезда, время пути, суммарное расстояние.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого. Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.



1. Анализ задачи

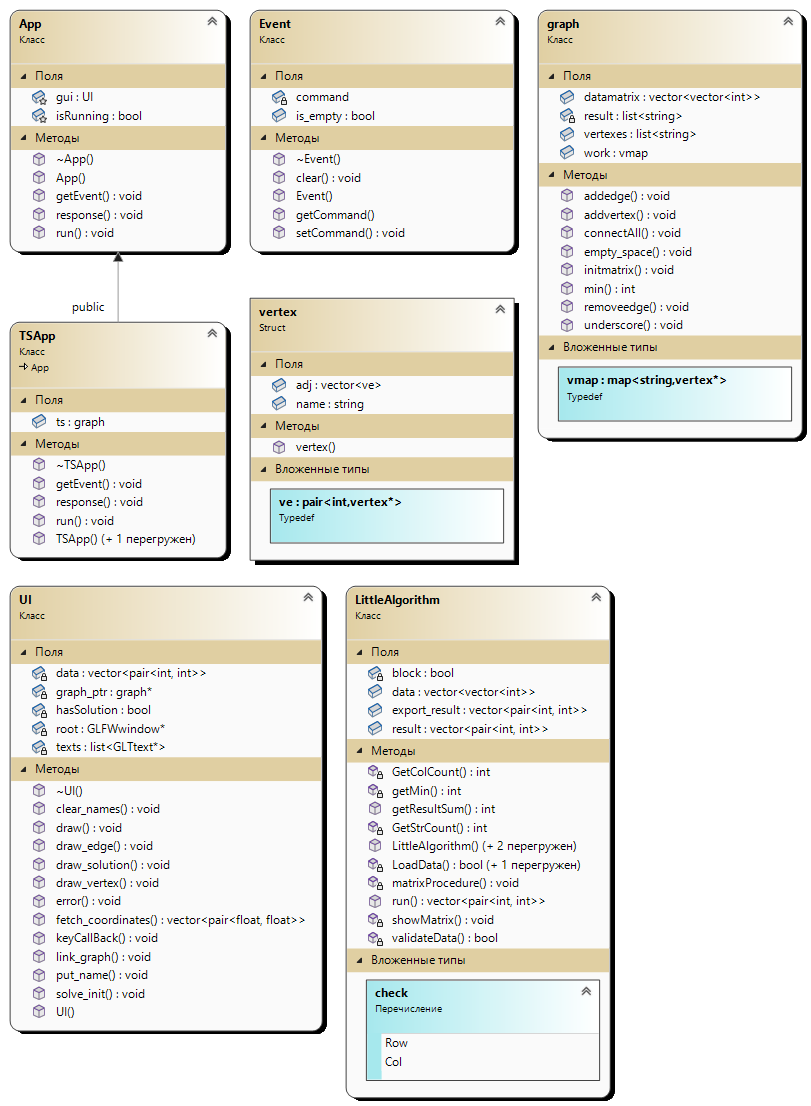
Задачу разобьём на составные части, которые реализуем в отдельных классах:

* LittleAlgorithm.h – метод ветвей и границ;
* UI.h – графика;
* App.h – цикл событий, работа с остальной частью проекта;
* Graph.h – работа с графами;

В файле main.cpp будет содержаться только запуск цикла событий.

Задача будет рекурсивно решаться методом ветвей и границ, т.е. методом Д. Литла. Графика будет реализована через функционал OpenGL и менеджера GLFW. Отрисовка текста будет реализована с применением библиотеки gltext.h.

1. Блок-схема



1. Код

App.h

#pragma once

#include "Event.h"

#include "Graph.h"

#include "UI.h"

// abstract class representing base App object

class App

{

protected:

bool isRunning;

UI gui;

public:

App() {}

~App() {}

virtual void run() = 0;

virtual void getEvent(Event&) = 0;

virtual void response(Event&) = 0;

};

// class representing Travelling Seller App

class TSApp : public App {

public:

graph ts;

TSApp();

TSApp(bool faster);

~TSApp();

void run() override;

void getEvent(Event&) override;

void response(Event&) override;

};

Event.h

#pragma once

const short int add\_edge = 1;

const short int remove\_edge = 2;

const short int find\_way = 3;

class Event

{

private:

short int command;

public:

bool is\_empty;

Event() { this->is\_empty = true; }

~Event() {};

void setCommand(int cmd) {

if ((cmd >= 0) && (cmd < 10)) {

this->is\_empty = false;

this->command = cmd;

}

}

short int getCommand() {

return this->command;

}

void clear() {

this->is\_empty = true;

this->command = -1;

}

};

Graph.h

#pragma once

#include <iostream>

#include <vector>

#include <map>

#include <string>

#include <list>

#include <fstream>

using namespace std;

struct vertex {

typedef pair<int, vertex\*> ve;

vector<ve> adj; //cost of edge, destination vertex

string name;

vertex(string s) : name(s) {}

};

class graph

{

list<string> result;

public:

vector<vector<int>> datamatrix;

typedef map<string, vertex\*> vmap; // dict of string and vertex (name: vertex)

list<string> vertexes;

vmap work;

void initmatrix();

void connectAll(const string&);

void addvertex(const string&);

void addedge(const string& from, const string& to, const double& cost);

void removeedge(const string&);

void empty\_space();

static void underscore(int q);

static int min(vector<int> pile);

};

class LittleAlgorithm {

public:

vector<vector<int>> data;

vector<pair<int, int>> result;

vector<pair<int, int>> export\_result;

LittleAlgorithm();

LittleAlgorithm(vector<vector<int>>);

LittleAlgorithm(char\*);

virtual vector<pair<int, int>> run();

int getResultSum(); // Считывание суммы всех выбранных дуг

private:

virtual bool validateData();

bool block = false;

enum check { Row, Col };

bool LoadData(char\*);

bool LoadData(vector<vector<int>>);

void matrixProcedure(vector<vector<int>>); // Метод в котором идет поиск цикла

void showMatrix(vector<vector<int>>); // Вывод матрицы

int getMin(vector<vector<int>>, int, check); // Поиск минимального элемента столбца/строки

int GetStrCount(ifstream&);

int GetColCount(ifstream&);

};

UI.h

#pragma once

#define GLFW\_INCLUDE\_NONE

#define \_USE\_MATH\_DEFINES

#include "graph.h"

#include <list>

#include <math.h>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#define GLT\_IMPLEMENTATION

#include "gltext.h"

class UI

{

graph\* graph\_ptr;

list <GLTtext\*> texts;

GLFWwindow\* root;

private:

bool hasSolution = false;

vector<pair<int, int>> data;

public:

UI();

~UI();

void link\_graph(graph\* data);

void draw();

void draw\_vertex(float x, float y);

void draw\_edge(float x1, float y1, float x2, float y2);

void draw\_solution();

void put\_name(string& name, GLfloat x, GLfloat y);

void clear\_names();

void solve\_init(vector<pair<int, int>>);

static void keyCallBack(GLFWwindow\* root, int key, int scancode, int action, int mode);

static void error(int code, const char\* desc);

static vector<pair<float, float>> fetch\_coordinates(pair<int, int>, int);

};

App.cpp

#include "App.h"

#include <iostream>

using namespace std;

TSApp::TSApp(){

gui.link\_graph(&this->ts);

this->isRunning = false;

}

TSApp::TSApp(bool faster) {

gui.link\_graph(&ts);

this->isRunning = faster;

}

TSApp::~TSApp() {

this->isRunning = false;

}

void TSApp::run(){

Event ev;

this->isRunning = true;

while (this->isRunning == true) {

cout << endl << "Travelling salesman problem solver. Pick an action:" << endl <<

"'1' - Add edge to graph and visualise it;" << endl <<

"'2' - Remove edge from graph and visualization;" << endl <<

"'3' - Find an optimal way to visit all verticles." << endl <<

"'9' - Render the graph" << endl <<

"'0' - Exit program" << endl << endl << ">>>";

this->getEvent(ev);

this->response(ev);

}

}

void TSApp::getEvent(Event& ev) {

int cmd;

cin >> cmd;

cout << endl;

ev.setCommand(cmd);

}

void TSApp::response(Event& ev) {

if (!ev.is\_empty) {

switch (ev.getCommand()) {

case add\_edge: {

string name;

cout << "Input edge name: ";

cin >> name;

this->ts.addvertex(name);

this->ts.connectAll(name);

break;

}

case remove\_edge: {

string name;

cout << "Input edge name: ";

cin >> name;

this->ts.removeedge(name);

break;

}

case find\_way:

{

ts.initmatrix();

LittleAlgorithm alg(ts.datamatrix);

vector<pair<int, int>> data = alg.run();

cout << ts.datamatrix[data[0].first-1][data[0].second-1];

int sum = ts.datamatrix[data[0].first - 1][data[0].second - 1];

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

sum += ts.datamatrix[data[i].first - 1][data[i].second - 1];

cout << " + " << ts.datamatrix[data[i].first - 1][data[i].second - 1];

}

cout << " = " << sum << endl;

gui.solve\_init(data);

break;

}

case 9:

gui.draw();

break;

case 0:

this->isRunning = false;

break;

}

}

ev.clear();

}

Graph.cpp

#include "graph.h"

#include <iomanip>

void graph::addvertex(const string& name)

{

vmap::iterator itr = work.find(name);

if (itr == work.end())

{

vertex\* v;

v = new vertex(name);

work[name] = v;

vertexes.push\_back(name);

return;

}

cout << "\nVertex already exists!";

}

void graph::addedge(const string& from, const string& to, const double& cost)

{

vertex\* f = (work.find(from)->second);

vertex\* t = (work.find(to)->second);

pair<int, vertex\*> edge = make\_pair(cost, t);

f->adj.push\_back(edge);

}

void graph::initmatrix()

{

cout << "adjacency matrix:" << endl;

datamatrix.resize(work.size());

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

datamatrix[i].resize(work.size());

}

list<string>::iterator it = vertexes.begin();

empty\_space();

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

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

datamatrix[i][j] = work[(\*it)]->adj[j].first;

}

if (work.find(\*it) != work.cend()) cout << " | " << setw(5) << "'" + work[(\*it)]->name + "'";

it++;

}

cout << endl; underscore(work.size()); cout << endl;

it = vertexes.begin();

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

cout << setw(5) << "'" + work[(\*it)]->name + "' | ";

it++;

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

if (datamatrix[i][j] != INT32\_MAX)

cout << setw(5) << left << datamatrix[i][j] << " | ";

else cout << setw(5) << left << "inf" << " | ";

}

if (datamatrix[i][work.size() - 1] != INT32\_MAX)

cout << setw(5) << left << datamatrix[i][work.size() - 1] << endl;

else cout << setw(5) << left << "inf" << endl;

}

}

void graph::connectAll(const string & name) {

for (list<string>::iterator it = vertexes.begin(); it != vertexes.end(); ++it) {

if (name != work[(\*it)]->name) {

int cost;

cout << "Input " << name << " -> " << work[(\*it)]->name << " cost: " << endl;

cin >> cost;

addedge(name, work[(\*it)]->name, cost);

addedge(work[(\*it)]->name, name, cost);

}

else {

addedge(name, work[(\*it)]->name, INT32\_MAX);

}

}

}

void graph::removeedge(const string& name) {

for (list<string>::iterator it = vertexes.begin(); it != vertexes.end();) {

if (work[(\*it)]->name == name) {

work.erase(\*it);

}

else {

++it;

}

}

}

int graph::min(vector<int> pile) {

int mini = 100000000;

for (int const& i : pile) {

if (i < mini && i != INT32\_MAX) mini = i;

}

return mini;

}

void graph::underscore(int q)

{

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

cout << "\_\_\_\_\_\_\_\_";

}

cout << "\_\_\_\_" << endl;

}

void graph::empty\_space() {

int max = 0;

for (auto const& it : work) {

if (it.second->name.size() > max) max = it.second->name.size();

}

cout << setw(max+2) << " ";

}

using namespace std;

LittleAlgorithm::LittleAlgorithm() {}

LittleAlgorithm::LittleAlgorithm(vector<vector<int>> Data) { LoadData(Data); }

LittleAlgorithm::LittleAlgorithm(char\* Path) { LoadData(Path); }

vector<pair<int, int>> LittleAlgorithm::run()

{

if (!validateData()) throw "Not valid data.";

matrixProcedure(vector<vector<int>>(data));

return export\_result;

}

int LittleAlgorithm::getMin(vector<vector<int>> matrix, int sel, check pos)

{

int min = INT32\_MAX;

for (int i = 0; i < matrix[sel].size() - 1; i++)

switch (pos)

{

case LittleAlgorithm::Row:

if (min > matrix[sel][i])

min = matrix[sel][i];

break;

case LittleAlgorithm::Col:

if (min > matrix[i][sel])

min = matrix[i][sel];

break;

}

return min;

}

void LittleAlgorithm::matrixProcedure(vector<vector<int>> matrix)

{

if (block) return;

//Определяем точку возврата и удаляем необходимое ребро

if (matrix.size() - 1 > 2) {

vector<int> vertexes;

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

vertexes.push\_back(result[i].first);

vertexes.push\_back(result[i].second);

}

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

pair<int, int> elem(INT32\_MAX, INT32\_MAX), elem1(INT32\_MAX, INT32\_MAX);

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

if (vertexes[i] != vertexes[j]) {

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

if (vertexes[i] == matrix[k][matrix[k].size() - 1]) elem.first = k;

if (vertexes[j] == matrix[k][matrix[k].size() - 1]) elem1.first = k;

}

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

if (vertexes[i] == matrix[matrix.size() - 1][k]) elem.second = k;

if (vertexes[j] == matrix[matrix.size() - 1][k]) elem1.second = k;

}

}

}

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

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

if (i == elem1.first && j == elem1.second)

matrix[elem1.first][elem1.second] = INT32\_MAX;

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

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

if (i == elem.first && j == elem.second)

matrix[elem.first][elem.second] = INT32\_MAX;

}

}

//Вычитаем из каждой строки минимальное значение

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

int min = 0;

if ((min = getMin(matrix, i, check::Row)) == INT32\_MAX) {

showMatrix(matrix);

cout << endl << "Bad road" << endl;

return;

}

if ((min = getMin(matrix, i, check::Row)) != 0)

for (int j = 0; j < matrix[i].size() - 1; j++)

if (matrix[i][j] != INT32\_MAX) matrix[i][j] -= min;

}

//Вычитаем из каждого столбца минимальное значение

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

int min = 0;

if ((min = getMin(matrix, i, check::Col)) == INT32\_MAX) {

showMatrix(matrix);

std::cout << endl << "Bad road" << endl;

return;

}

if ((min = getMin(matrix, i, check::Col)) != 0)

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

if (matrix[j][i] != INT32\_MAX) matrix[j][i] -= min;

}

//Находим максимально оцененный ноль

int Max = 0;

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

for (int j = 0; j < matrix[i].size() - 1; j++)

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

matrix[i][j] = INT32\_MAX;

int max = (getMin(matrix, i, check::Row) == INT32\_MAX || getMin(matrix, j, check::Col) == INT32\_MAX) ? INT32\_MAX : getMin(matrix, i, check::Row) + getMin(matrix, j, check::Col);

if (max > Max) Max = max;

matrix[i][j] = 0;

}

//Находим все нули максимальная оценка которых равна Max

vector<pair<int, int>> Maxs;

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

for (int j = 0; j < matrix[i].size() - 1; j++)

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

matrix[i][j] = INT32\_MAX;

int max = (getMin(matrix, i, check::Row) == INT32\_MAX || getMin(matrix, j, check::Col) == INT32\_MAX) ? INT32\_MAX : getMin(matrix, i, check::Row) + getMin(matrix, j, check::Col);

if (max == Max) Maxs.push\_back(pair<int, int>(matrix[i][matrix.size() - 1], matrix[matrix.size() - 1][j]));

matrix[i][j] = 0;

}

//Debug: Вывод координат выбранных нулей

//std::cout << "Maxs - ";

//for (int i = 0; i < Maxs.size(); i++)

//std::cout << Maxs[i].first << " " << Maxs[i].second << "\t";

//std::cout << endl;

//Вывод матрицы

showMatrix(matrix);

//Завершаем выполнение данной ветви если нету нулей

if (Maxs.size() == 0) {

std::cout << "Bad road." << endl;

return;

}

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

//Добавляем вершину в массив с результатом

result.push\_back(Maxs[i]);

//Если размер матрицы порядка 1, выводим результат и завершаем текущию ветвь

if (matrix.size() - 1 == 1) {

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

std::cout << "(" << result[i].first << ", " << result[i].second << ")\t";

std::cout << endl;

if (export\_result.empty()) {

export\_result = result;

block = true;

}

result.pop\_back();

return;

}

//Создаем копию текущей матрицы и удаляем из нее строку и столбец выбранного нуля

vector<vector<int>> temp(matrix);

pair<int, int> elem(INT32\_MAX, INT32\_MAX), elem1(INT32\_MAX, INT32\_MAX);

for (int j = 0; j < temp[temp.size() - 1].size() - 1; j++) {

if (Maxs[i].first == temp[j][temp[j].size() - 1]) elem.first = j;

if (Maxs[i].second == temp[j][temp[j].size() - 1]) elem1.first = j;

}

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

if (Maxs[i].second == temp[temp.size() - 1][j]) elem.second = j;

if (Maxs[i].first == temp[temp.size() - 1][j]) elem1.second = j;

}

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

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

if (i == elem1.first && j == elem1.second)

temp[elem1.first][elem1.second] = INT32\_MAX;

for (int j = 0; j < temp[temp.size() - 1].size(); j++)

temp[j].erase(temp[j].begin() + elem.second);

temp.erase(temp.begin() + elem.first);

//Вызываем рекурсивно эту же функцию для уже новой матрицы

matrixProcedure(temp);

//Удаляем последние значение из массива с результатом

result.pop\_back();

}

}

//Вывод матрицы

void LittleAlgorithm::showMatrix(vector<vector<int>> temp)

{

std::cout << endl;

std::cout << "\t";

for (int i = 0; i < temp[temp.size() - 1].size() - 1; i++) {

std::cout << temp[temp.size() - 1][i] << "\t";

}

std::cout << endl;

for (int i = 0; i < temp[0].size()+1; i++)

for (int j = 0; j < 6; j++) std::cout << "\_";

std::cout << endl << endl;

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

std::cout << temp[i][temp.size() - 1] << " | " << "\t";

for (int j = 0; j < temp[i].size() - 1; j++)

if (temp[i][j] != INT32\_MAX && j != temp.size() - 1)std::cout << temp[i][j] << "\t";

else std::cout << "inf" << "\t";

std::cout << endl;

}

std::cout << endl;

}

int LittleAlgorithm::getResultSum()

{

int sum = 0;

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

sum += data[result[i].first - 1][result[i].second - 1];

return sum;

}

bool LittleAlgorithm::validateData()

{

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

for (int j = 0; j < data[i].size(); j++)

if (data[i][j] == 0)

data[i][j] = INT32\_MAX;

vector<vector<int>> temp(data);

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

data[i].push\_back(i + 1);

vector<int> numeration;

for (int i = 0; i < data[0].size(); i++)

numeration.push\_back(i + 1);

data.push\_back(numeration);

return true;

}

bool LittleAlgorithm::LoadData(char\* Path)

{

ifstream file(Path, ios\_base::in);

if (!file.is\_open()) throw "Can't open data file";

int rows = GetStrCount(file), cols = GetColCount(file);

for (int i = 0; i < rows; i++)

{

vector<int> temp;

for (int j = 0; j < cols; j++)

{

int n;

file >> n;

temp.push\_back(n);

}

data.push\_back(vector<int>(temp));

}

file.seekg(0);

return true;

}

bool LittleAlgorithm::LoadData(vector<vector<int>> Data)

{

data = Data;

return true;

}

int LittleAlgorithm::GetStrCount(ifstream& file)

{

int count = 0;

while (!file.eof()) {

file.getline(new char[255], 255);

count++;

};

file.seekg(0);

return count;

}

int LittleAlgorithm::GetColCount(ifstream& file)

{

int count = 0;

char str[255];

file.getline(str, 255);

for (int i = 0; i < strlen(str); i++)

{

if (isdigit(str[i]))

if (i != 0 && !isdigit(str[i - 1]))

count++;

else if (i == 0)

count++;

}

file.seekg(0);

return count;

}

UI.cpp

#include "UI.h"

#include "Windows.h"

void UI::error(int code, const char\* desc) {

fputs(desc, stderr);

}

void UI::link\_graph(graph\* data)

{

this->graph\_ptr = data;

}

void UI::keyCallBack(GLFWwindow\* root, int key, int scancode, int action, int mode) {

int state = glfwGetKey(root, GLFW\_KEY\_ESCAPE);

if (state == GLFW\_PRESS) {

glfwSetWindowShouldClose(root, GLFW\_TRUE);

}

}

UI::UI() {

}

UI::~UI() {

}

void UI::draw() {

if (!glfwInit()) delete this;

glfwSetErrorCallback(error);

root = glfwCreateWindow(900, 900, "Travelling Merchant by buks.net", NULL, NULL);

if (root == NULL) delete this;

glfwMakeContextCurrent(root);

glfwSwapInterval(1);

glfwSetKeyCallback(root, keyCallBack);

glClearColor(255, 255, 255, 1);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glColor3f(1.f, 1.f, 1.f);

glewExperimental = GL\_TRUE;

if (glewInit() != GLEW\_OK) throw runtime\_error("glewInit failed");

if (!gltInit()) cout << "Font init error";

while (!glfwWindowShouldClose(root)) {

int width, height;

glfwGetFramebufferSize(root, &width, &height);

glClear(GL\_COLOR\_BUFFER\_BIT);

clear\_names();

glLineWidth(1.f);

glViewport(0, 0, width, height);

int i = 0, j = 0;

float theta, x, y, x1, y1, x2, y2;

for (auto const& it : (\*graph\_ptr).work) {

theta = 2.0f \* 3.1415926f \* float(i - 1) / float((\*graph\_ptr).work.size());

x = 0.55 \* cosf(theta);

y = 0.55 \* sinf(theta);

theta = 2.0f \* 3.1415926f \* float(i) / float((\*graph\_ptr).work.size());

x1 = 0.55 \* cosf(theta);

y1 = 0.55 \* sinf(theta);

draw\_edge(x + 0.01, y + 0.125, x1 + 0.01, y1 + 0.125);

for (auto const& edge : it.second->adj) {

if (j > 0 && j < (\*graph\_ptr).work.size()-1) {

theta = 2.0f \* 3.1415926f \* float(j) / float((\*graph\_ptr).work.size());

x2 = 0.55 \* cosf(theta);

y2 = 0.55 \* sinf(theta);

if (edge.second->name != it.second->name) draw\_edge(x + 0.01, y + 0.125, x2 + 0.01, y2 + 0.125);

}

j++;

}

j = 0;

i++;

}

i = 0;

for (auto const& it : (\*graph\_ptr).work) {

theta = 2.0f \* 3.1415926f \* float(i - 1) / float((\*graph\_ptr).work.size());

x = 0.55 \* cosf(theta);

y = 0.55 \* sinf(theta);

theta = 2.0f \* 3.1415926f \* float(i) / float((\*graph\_ptr).work.size());

x1 = 0.55 \* cosf(theta);

y1 = 0.55 \* sinf(theta);

draw\_vertex(x, y);

i++;

}

i = 0;

for (auto const& it : (\*graph\_ptr).work) {

theta = 2.0f \* 3.1415926f \* float(i - 1) / float((\*graph\_ptr).work.size());

x = 0.55 \* cosf(theta);

y = 0.55 \* sinf(theta);

theta = 2.0f \* 3.1415926f \* float(i) / float((\*graph\_ptr).work.size());

x1 = 0.55 \* cosf(theta);

y1 = 0.55 \* sinf(theta);

j = 0;

put\_name(it.second->name, x, y);

i++;

}

if (hasSolution) draw\_solution();

Sleep(0.1);

glfwSwapBuffers(root);

glfwPollEvents();

}

gltTerminate();

glfwDestroyWindow(this->root);

glfwTerminate();

}

void UI::draw\_vertex(float x, float y) {

//circle

glBegin(GL\_POLYGON);

glColor3f(.9098f, .8941f, .00785f);

float theta;

for (int i = 0; i < 100; i++) {

theta = 2.0f \* 3.1415926f \* float(i) / float(100);//get the current angle

x += 0.008 \* cosf(theta);//calculate the x component

y += 0.008 \* sinf(theta);//calculate the y component

glVertex2f(x, y);//output vertex

}

glEnd();

//border

glBegin(GL\_LINE\_LOOP);

glColor3f(0.f, 0.f, 0.f);

for (int i = 0; i < 100; i++) {

float theta = 2.0f \* 3.1415926f \* float(i) / float(100);

x += 0.008 \* cosf(theta);//calculate the x component

y += 0.008 \* sinf(theta);//calculate the y component

glVertex2f(x, y);

}

glEnd();

}

void UI::draw\_edge(float x1, float y1, float x2, float y2) {

glColor3f(0.f, 0.f, 0.f);

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

void UI::put\_name(string& name, GLfloat x, GLfloat y) {

bool f = false;

for (auto const& gr : (\*graph\_ptr).work) {

if (gr.second->adj.size()!=0) f = true;

}

if (f) {

GLTtext\* text = gltCreateText();

gltSetText(text, name.c\_str());

texts.push\_back(text);

gltBeginDraw();

gltColor(1.f, 1.f, 1.f, 1.f);

gltDrawText2D(text, (x + 0.95) \* 450, (y + 0.85) \* 450, 1);

gltEndDraw();

}

}

void UI::clear\_names() {

for (auto text : texts) {

gltDeleteText(text);

}

texts.clear();

}

void UI::solve\_init(vector<pair<int, int>> arg)

{

data = arg;

hasSolution = true;

}

vector<pair<float, float>> UI::fetch\_coordinates(pair<int, int> coord, int q) {

float theta = 2.0f \* 3.1415926f \* float(coord.first - 1) / float(q), x, y, x1, y1;

vector < pair<float, float>> result;

pair<float, float> one, two;

one.first = 0.55 \* cosf(theta);

one.second = 0.55 \* sinf(theta);

theta = 2.0f \* 3.1415926f \* float(coord.second-1) / float(q);

two.first = 0.55 \* cosf(theta);

two.second = 0.55 \* sinf(theta);

result.push\_back(one);

result.push\_back(two);

return result;

}

void UI::draw\_solution()

{

vector<pair<float, float>> coord;

for (vector<pair<int, int>>::iterator it = data.begin(); it != data.end(); it++) {

glColor3f(0.f, 0.5f, 0.f);

glLineWidth(5.f);

glBegin(GL\_LINES);

coord = fetch\_coordinates((\*it), data.size());

glVertex2f(coord[0].first-0.02, coord[0].second+0.18);

glVertex2f(coord[1].first-0.02, coord[1].second+0.18);

glEnd();

}

}

Main.cpp

#include "App.h"

#include <iostream>

using namespace std;

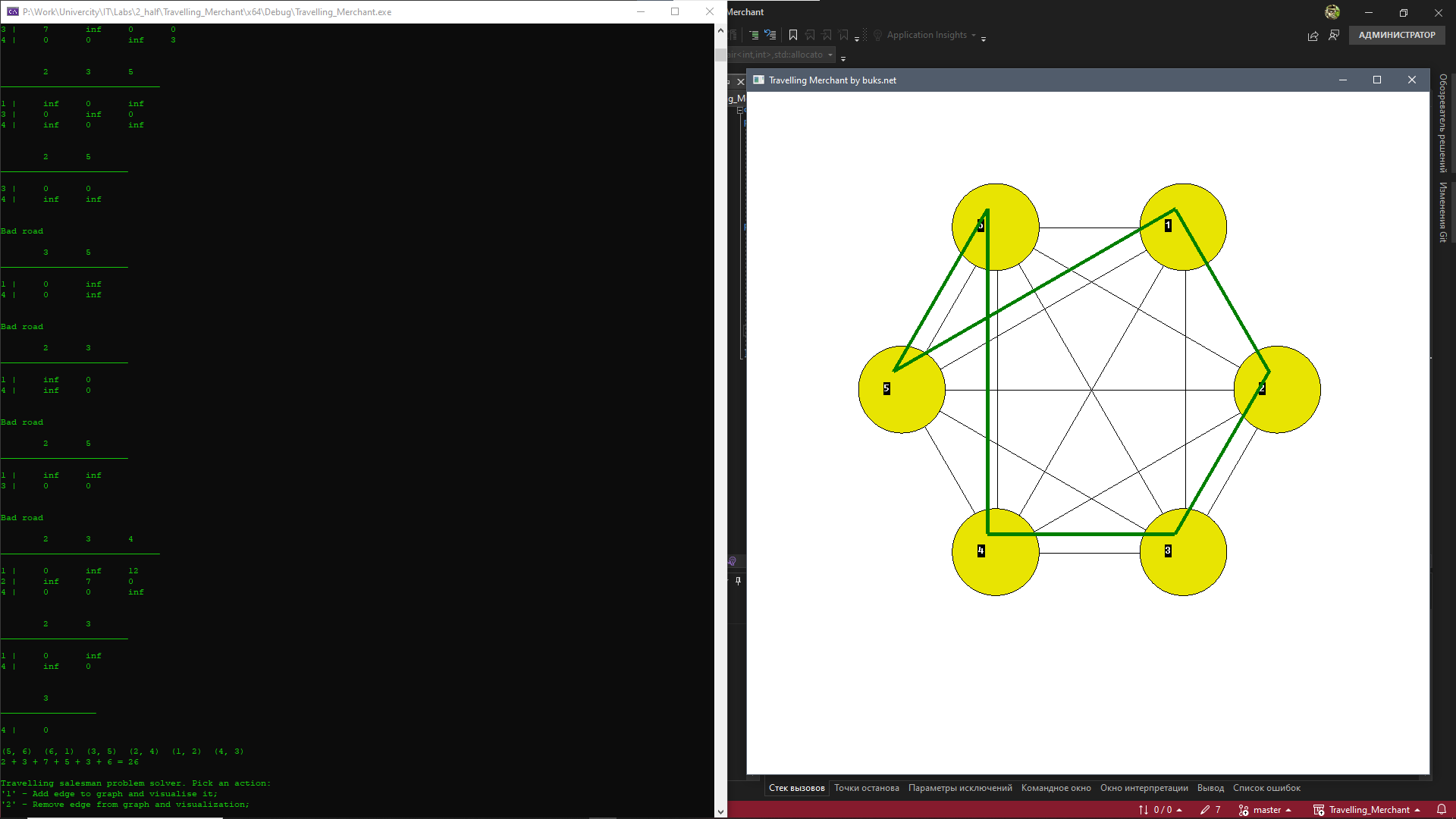
int main() {

TSApp app;

app.run();

}

1. Вывод



1. GitHub

<https://github.com/buksnet/Travelling_Merchant>