Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра **«**Информационные технологии и автоматизированные системы**»**

**ОТЧЕТ**

**ПО ТВОРЧЕСКОЙ РАБОТЕ №2**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Задача Коммивояжёра

Выполнил работу

Студент группы РИС-22-1Б

Лихачев Д.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Введение**

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

**Постановка задачи**

* Реализация приложения на основе OpenGL, для решения задачи Коммивояжёра.
* Изучить метод ветвей и границ.
* Визуализация графа и решения задачи.

Метод ветвей и границ метод основан на разбиении исходной задачи на более мелкие подзадачи, а затем применении оценок верхней и нижней границ, чтобы исключить из рассмотрения часть возможных решений, которые не могут быть оптимальными

**Код программы**

**Graph.h**

#pragma once

#include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

int n;

int\*\* s;

int\* way;

int\*\*\* Matrix\_new;

int\* path;

bool pr = false;

vector<int> CommiWays;

int WinW;

int WinH;

int R;

struct Node\_Coordinatios

{

int x, y;

};

Node\_Coordinatios Node\_Coord[15];

template<class T>

class Graph

{

vector<T> Node\_List;

public:

int Matrix[15][15];

Graph() {

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 15; j++) {

this->Matrix[i][j] = 0;

}

}

}

void Adding\_Node(const T& node) {

if (this->Node\_List.size() == 15) {

cout << "\nК сожалению, вы хотите ввести для подсчета более 15 городов, что противоречит условиям конкретной задачи";

return;

}

else {

this->Node\_List.push\_back(node);

}

}

void Deleting\_Node() {

if (this->Node\_List.size() == 0) {

cout << "\nК сожалению, городов в списке больше нет: удалять нечего";

}

else {

this->Node\_List.pop\_back();

}

}

int Number(const T& node) {

for (int i = 0; i < this->Node\_List.size(); i++) {

if (this->Node\_List[i] == node) {

return i;

}

}

return -1;

}

void Adding\_Way(const T& node\_fst, const T& node\_snd, const int len) {

int Number\_1, Number\_2;

if (this->Number(node\_fst) == -1) {

cout << "\nОшибка. Города с номером " << node\_snd << " не существует\n";

}

else {

if (this->Number(node\_snd) == -1) {

cout << "\nОшибка. Города с номером " << node\_snd << " не существует\n";

}

else {

Number\_1 = Number(node\_fst);

Number\_2 = Number(node\_snd);

if (Matrix[Number\_1][Number\_2] == 0 && Matrix[Number\_2][Number\_1] == 0) {

this->Matrix[Number\_1][Number\_2] = len;

this->Matrix[Number\_2][Number\_1] = len;

}

else {

cout << "\nМежду этими городами уже проложен путь";

}

}

}

}

void Matrix\_Output() {

if ((this->Node\_List.size()) == 0) {

cout << "\nСписок городов пуст. Матрица смежности графа не может быть выведена";

}

else {

cout << "\nМатрица смежности городского графа:\n";

cout << "\t";

for (int i = 0; i < Node\_List.size(); i++) {

cout << Node\_List[i] << "\t|\t";

}

cout << "\n\n";

for (int i = 0; i < Node\_List.size(); i++) {

cout << Node\_List[i] << "\t";

for (int j = 0; j < Node\_List.size(); j++) {

cout << this->Matrix[i][j] << "\t|\t";

}

cout << "\n\n";

}

}

}

void Graph\_Output();

int GetVertPos(const int& vertex);

~Graph() {};

};

int Node\_Quantity = 0;

int Way\_Quantity = 0;

Graph<int> graph;

void Finding\_Answer(int\*\*\* Matrix\_new, int n, int\*\* Help\_Matrix, int\* path) {

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

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

int min = 1000000;

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

if (Matrix\_new[i][j] && min > \*Matrix\_new[i][j])

min = \*Matrix\_new[i][j];

}

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

if (Matrix\_new[i][j])

\*Matrix\_new[i][j] -= min;

}

}

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

int min = 1000000;

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

if (Matrix\_new[i][j] && min > \*Matrix\_new[i][j])

min = \*Matrix\_new[i][j];

}

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

if (Matrix\_new[i][j])

\*Matrix\_new[i][j] -= min;

}

}

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

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

Help\_Matrix[i][j] = 0;

}

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

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

if (Matrix\_new[i][j] && !\*Matrix\_new[i][j]) {

int col\_min = 1000000;

int string\_min = 1000000;

for (int l = 0; l < n; l++)

if (l != i && Matrix\_new[l][j] && col\_min > \*Matrix\_new[l][j])

col\_min = \*Matrix\_new[l][j];

for (int l = 0; l < n; l++)

if (l != j && Matrix\_new[i][l] && string\_min > \*Matrix\_new[i][l])

string\_min = \*Matrix\_new[i][l];

Help\_Matrix[i][j] = col\_min + string\_min;

}

}

int mcost = 0, mi = 0, mj = 0;

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

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

if (Matrix\_new[i][j] && mcost < Help\_Matrix[i][j])

{

mcost = Help\_Matrix[i][j];

mi = i;

mj = j;

}

path[mi] = mj;

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

Matrix\_new[i][mj] = nullptr;

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

Matrix\_new[mi][i] = nullptr;

Matrix\_new[mj][mi] = nullptr;

}

}

void Matrix\_Creation(int\*\*\*& Matrix\_new, int& n, int\*\*& s, int\*& way) {

n = Node\_Quantity;

s = new int\* [n];

way = new int[n];

Matrix\_new = new int\*\* [n];

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

s[i] = new int[n];

}

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

Matrix\_new[i] = new int\* [n];

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

if (graph.Matrix[i][j] == 0) {

Matrix\_new[i][j] = nullptr;

continue;

}

Matrix\_new[i][j] = new int(graph.Matrix[i][j]);

}

}

}

void Kommivoyager(int\*\*\* Matrix\_new, int n, int\*\* s, int\* way)

{

CommiWays.clear();

Matrix\_Creation(Matrix\_new, n, s, way);

Finding\_Answer(Matrix\_new, n, s, way);

int sum\_length = 0;

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

j = way[i];

sum\_length += graph.Matrix[i][j];

}

cout << "\n\nКратчайший путь, пройденный Коммивояжером ";

int slen = 0;

for (int l = 0; l < n;) {

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

if ((slen == 0) || (slen == i + 1)) {

if (slen == 0) {

CommiWays.push\_back(i+1);

cout << i + 1;

}

j = way[i];

slen = j + 1;

if (slen > 0) {

cout << " -> " << slen;

CommiWays.push\_back(slen);

}

l++;

}

}

}

cout << "\n\nОбщая длина пути, который пройдет Коммивояжер: " << sum\_length << "\n\n";

if (pr)

pr = false;

else

pr = true;

}

void Ouput\_Ways(int\*\*\* Matrix\_new, int n, int\*\* s, int\* way) {

Matrix\_Creation(Matrix\_new, n, s, way);

Finding\_Answer(Matrix\_new, n, s, way);

int sum\_length = 0;

cout << "\nВсе пути между городами, по которым пройдет коммивояжер: \n";

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

j = way[i];

cout << i + 1 << " -> " << j + 1 << '\n';

}

}

Graph <int> Create\_Graph() {

Graph<int> graph;

int first, second, length;

int\* first\_Node;

int\* second\_Node;

int\* New\_Node;

cout << "\nВведите количество городов, которые необходимо посетить Коммивояжеру : ";

cout << "\n";

cin >> Node\_Quantity;

cout << "\nВведите количество маршрутов между этими городами: ";

cout << "\n";

cin >> Way\_Quantity;

for (int i = 1; i <= Node\_Quantity; ++i) {

New\_Node = &i;

graph.Adding\_Node(\*New\_Node);

}

for (int i = 0; i < Way\_Quantity; ++i) {

cout << "\nВведите номер города, из которого начинается маршрут: ";

cin >> first;

first\_Node = &first;

cout << "\nВведите номер города, к которому ведет маршрут: ";

cin >> second;

second\_Node = &second;

cout << "\n\tВведите расстояние между городами " << first << " и " << second << " : ";

cin >> length;

graph.Adding\_Way(\*first\_Node, \*second\_Node, length);

}

return graph;

}

void save\_node(int i, int n)

{

int New\_R;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

New\_R = WinH / 2 - R - 10;

}

else {

R = 5 \* (WinW / 13) / n;

New\_R = WinW / 2 - R - 10;

}

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

float y1 = New\_R \* cos(theta) + y0;

float x1 = New\_R \* sin(theta) + x0;

Node\_Coord[i].x = x1;

Node\_Coord[i].y = y1;

}

void Draw\_City(int x, int y, int R)

{

glColor3f(1.0 / 255.0 \* 156, 1.0 / 255.0 \* 156, 1.0 / 255.0 \* 156); //перламутровый светло серый

float x1, y1;

glBegin(GL\_POLYGON);

for (int i = 0; i < 360; i++) {

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R / 2 \* cos(theta) + y;

x1 = R / 2 \* sin(theta) + x;;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0f, 0.0f, 0.0f);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 360; i++) {

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R / 2 \* cos(theta) + y;

x2 = R / 2 \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void New\_Text(int nom, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(nom);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void Draw\_Node(int n)

{

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

Draw\_City(Node\_Coord[i].x, Node\_Coord[i].y, R);

New\_Text(i + 1, Node\_Coord[i].x, Node\_Coord[i].y);

}

}

void Draw\_Way(int text, int x0, int y0, int x1, int y1, bool b)

{

if (b)

glColor3f(1.0f, 0.0f, 0.0f);

else

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

if (b) {

New\_Text(text, (x0 + x1) / 2 - 10, (y0 + y1) / 2 - 10);

}

else {

New\_Text(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);

}

}

template<class T>

void Graph<T>::Graph\_Output()

{

for (int i = 0; i < Node\_List.size(); i++) {

save\_node(i, Node\_List.size());

}

for (int i = 0; i < Node\_List.size(); i++) {

for (int j = i + 1; j < Node\_List.size(); j++) {

if (Matrix[i][j] != 0) {

Draw\_Way(Matrix[i][j], Node\_Coord[i].x, Node\_Coord[i].y, Node\_Coord[j].x, Node\_Coord[j].y, false);

}

}

}

Draw\_Node(Node\_List.size());

}

void reshape(int w, int sum\_length) {

WinW = w;

WinH = sum\_length;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void drawMenuText(string text, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = text;

glRasterPos2i(x1 + 5, y1 - 20);

for (int j = 0; j < s.length(); j++)

glutBitmapCharacter(font, s[j]);

}

void drawMenu() {

int width = 60;

int lgth = 730;

glColor3d(1.0,1.0,1.0);

glBegin(GL\_QUADS);

glVertex2i(width - 20, lgth - width - 270);

glVertex2i(width + 270, lgth - width - 270);

glVertex2i(width + 270, lgth - width + 75);

glVertex2i(width - 20, lgth - width + 75);

glEnd();

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 30);

glVertex2i(width + 135, lgth - width - 30);

glVertex2i(width + 135, lgth - width);

glVertex2i(width, lgth - width);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Number of ways ", width - 2, lgth - width + 50);

glColor3d(1.0, 1.0, 1.0);

drawMenuText("(increase / decrease): ", width - 2, lgth - width + 25);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 30);

glVertex2i(width + 150, lgth - width - 30);

glVertex2i(width + 150, lgth - width);

glVertex2i(width, lgth - width);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("+", width + 60, lgth - width - 2);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 70);

glVertex2i(width + 150, lgth - width - 70);

glVertex2i(width + 150, lgth - width - 40);

glVertex2i(width, lgth - width - 40);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("-", width + 60, lgth - width - 42);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 110);

glVertex2i(width + 255, lgth - width - 110);

glVertex2i(width + 255, lgth - width - 80);

glVertex2i(width, lgth - width - 80);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Output the adjacency matrix", width, lgth - width - 82);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 180);

glVertex2i(width + 255, lgth - width - 180);

glVertex2i(width + 255, lgth - width - 120);

glVertex2i(width, lgth - width - 120);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText("Calculate the path traveled by ", width, lgth - width - 122);

glColor3d(1.0, 1.0, 1.0);

drawMenuText("a traveling salesman", width + 50, lgth - width - 150);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 220);

glVertex2i(width + 255, lgth - width - 220);

glVertex2i(width + 255, lgth - width - 190);

glVertex2i(width, lgth - width - 190);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" Create a new road map", width, lgth - width - 190);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_QUADS);

glVertex2i(width, lgth - width - 230);

glVertex2i(width + 255, lgth - width - 230);

glVertex2i(width + 255, lgth - width - 260);

glVertex2i(width, lgth - width - 260);

glEnd();

glColor3d(1.0, 1.0, 1.0);

drawMenuText(" List of possible ways", width, lgth - width - 230);

}

void mouseClick(int button, int stat, int x, int y) {

int width = 60;

int lgth = 730;

if (stat == GLUT\_DOWN) {

if (x > width && x < width + 135 && y > width && y < width + 30) {

int first;

int second;

int length;

int\* first\_node;

int\* second\_node;

cout << "\nНомер добавляемого города: ";

cin >> first;

first\_node = &first;

cout << "\nНомер города, с которым будет связан: ";

cin >> second;

cout << "\n\n";

second\_node = &second;

if (first > Node\_Quantity || second > Node\_Quantity) {

Node\_Quantity++;

int\* New\_Node = &Node\_Quantity;

graph.Adding\_Node(\*New\_Node);

}

cout << "Длина пути между городами: "; cin >> length; cout << endl;

graph.Adding\_Way(\*first\_node, \*second\_node, length);

}

if (x > width && x < width + 135 && y > width + 40 && y < width + 70) {

int first;

int second;

int length;

int\* first\_node;

cout << "\nНомер удаляемого города: ";

cin >> first;

cout << "\n";

first\_node = &first;

if (first == Node\_Quantity) {

Node\_Quantity--;

graph.Deleting\_Node();

}

else cout << "\nУдалить это город невозможно \n";

}

if (x > width && x < width + 240 && y > width + 80 && y < width + 100) {

graph.Matrix\_Output();

}

if (x > width && x < width + 240 && y > width + 120 && y < width + 140) {

Kommivoyager(Matrix\_new, n, s, way);

}

if (x > width && x < width + 240 && y > width + 160 && y < width + 200) {

graph = Create\_Graph();

}

if (x > width && x < width + 240 && y > width + 230 && y < width + 260) {

Ouput\_Ways(Matrix\_new, n, s, way);

}

}

glutPostRedisplay();

}

template<class T>

int Graph<T>::GetVertPos(const int& vertex)

{

for (size\_t i = 0; i < Node\_List.size(); i++)

if (Node\_List[i] == vertex)

return i;

return -1;

}

void display() {

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.Graph\_Output();

if (pr)

{

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

{

int vertPos1 = graph.GetVertPos(CommiWays[i]);

int vertPos2 = graph.GetVertPos(CommiWays[i+1]);

Draw\_Way(i + 1, Node\_Coord[vertPos1].x, Node\_Coord[vertPos1].y, Node\_Coord[vertPos2].x, Node\_Coord[vertPos2].y, true);

}

}

drawMenu();

glutSwapBuffers();

}

**Source1.cpp**

#include "graph.h"

#include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <vector>

#include <sstream>

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

{

setlocale(LC\_ALL, "RUS");

glutInit(&argc, argv);

cout << "Вашему вниманию представлена задача Коммивояжера. \nЧтобы программа работала корректно, введите не более 14 различных городов, через которые вам нужно проложить маршрут. \nДля удобства заполнения запишите каждый город в соответствии с его серийным номером. \n\n\n";

graph = Create\_Graph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

// Setting interface window options

glutInitWindowSize(1350, 730);

glutCreateWindow("Коммивояжер");

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouseClick);

glutMainLoop();

return 0;

}

**Вывод программы**



Вывод решения Коммивояжера:



***Рисунок 1 – вывод программы***



***Рисунок 2 – визуализация графа***



***Рисунок 3 – визуализация решения задачи***

**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.