Министерство образование Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

Выполнила:

студентка группы 20ВВ1:

Зацепилина Е.С.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза 2021 г.

**Цель работы**: освоить алгоритм поиска расстояний для разных видов графов в матричном представлении, освоить способ модернизации программы, с помощью которого можно запускать её с параметрами командной строки.

**Задание на лабораторную работу:**

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс queue из стандартной библиотеки С++.

3.\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа G. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

Задание 2\*

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже). В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Ход выполнения задания:**

**1. Выбор вида графа и генерация матрицы смежности.**

Для выполнения задания на лабораторную работу были написана программа 7 на языке С++ в среде Visual Studio 2019. Полный текст программы смотрите в п. «Листинг».

При запуске программа принимает два аргумента:

1) -or/-nor – ориентированный/неориентированный граф;

2) -vz/-nvz – взвешенный/невзвешенный граф.

В случае, если аргументов передано больше или меньше, или аргументы введены неправильно, то будут выведены ошибки и программу нужно будет запустить заново. Если аргументы введены верно, тогда переменным логического типа, определяющим вид графа, будут присвоены соответствующие значения:

if (argc != 3) {

cout << "Количество аргументов больше или меньше двух" << endl;

return 0;

}

if (strcmp(argv[1], "-nor") != 0) {

if (strcmp(argv[1], "-or") != 0) {

cout << "Первый аргумент указан не верно" << endl;

return 0;

}

}

if (strcmp(argv[2], "-nvz") != 0) {

if (strcmp(argv[2], "-vz") != 0) {

cout << "Второй аргумент указан не верно" << endl;

return 0;

}

}

if (strcmp(argv[1], "-or") == 0) orient = 1;

else orient = 0;

if (strcmp(argv[2], "-vz") == 0) vzves = 1;

else vzves = 0;

Затем осуществляется ввод кол-ва вершин n и инициализация матрицы смежности с помощью генератора случайных чисел. Во вложенном цикле с установленным числом повторений от 0 до n осуществляется инициализация элементов матрицы. Сначала определяется, находится ли элемент матрицы на главной диагонали матрицы. Если да, то элементу присваивается 0. Также проверяется ориентированность графа. Если он неориентированный, то цикл прерывается. Если элемент матрицы не находится на главной диагонали, то ему присваивается значение 0 или значение от 30 до 100, если граф взвешенный. В ином случае присваивается 0 или 1 с шансом 30 к 70 соответственно. Затем проверяется ориентированность графа: если граф неориентированный, то симметричному элементу присваивается то же значение.

int n;

printf("Введите кол-во вершин: ");

cin >> n;

m.resize(n);

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

m.at(i).resize(n);

}

srand(time(NULL));

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

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

if (i == j) {

m[i][j] = 0;

if (!orient) break;

}

else {

int z = rand() % 101;

if (!vzves) {

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = 1; }

}

else {

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = z; }

}

if (!orient) m[j][i] = m[i][j];

}

}

}

**2. Осуществление поиска расстояний.**

В функцию поиска расстояний BFSD (vector <vector <int>> gr, int v) передается два параметра, где gr – матрица смежности графа, а v – индекс вершины, для которой нужно найти расстояния. Поиск расстояний начинается с инициализации пустой очереди. Затем в неё добавляется индекс вершины, для которой осуществляем обход (первой в очередь попадет вершина, для которой ищем расстояния). Эту вершину помечаем как посещенную. Затем начинает свою работу цикл, который работает пока очередь не пуста. В переменную v записывается элемент который в очереди записан первым, из очереди удаляется первый элемент. Затем начинает свою работу цикл с установленным числом повторений для каждой вершины. Если расстояние до просматриваемой вершины меньше чем сумма расстояний до вершины v и расстояния из v до просматриваемой вершины и если существует путь от v до просматриваемой вершины, то в очередь добавляется эта вершина и расстояние до просматриваемой вершины обновляется на сумму.

void BFSD(vector <vector <int>> gr, int v) {

queue <unsigned int> q = {};

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

printf(" %d", v + 1);

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

if (gr[v][i] > 0 && DIST[i] > DIST[v] + gr[v][i]) {

q.push(i);

DIST[i] = DIST[v] + gr[v][i];

}

}

}

}

**Вывод:** в результате выполнения лабораторной работы были получены навыки реализации поиска расстояний в различных видах графов. Также была проведена модернизация программы, которая позволила запускать программу с параметрами командной строки.

**Листинг:**

**7.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <windows.h>

#include <vector>

#include <queue>

using namespace std;

vector <int> DIST;

void vivod\_m(vector <vector<int>> arr) {//Функция вывода одной матрицы

cout << (" |");

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

printf("%-4d|", i + 1);

}

cout << ("\n");

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

printf("|");

printf("%-2d|", i + 1);

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

printf("%-4d|", arr[i][j]);

}

printf("\n");

}

}

void udalenie(vector <vector<int>>& arr, int v) {//Удаление вершины из матрицы

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

auto iter = arr[i].cbegin();

arr[i].erase(iter + v - 1);

}

auto iter = arr.cbegin();

arr.erase(iter + v - 1);

}

void clear(vector <vector<int>>& arr) {//Удаление всех вершин, у которых отсутвуют рёбра (отчистка матрицы)

int val;

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

val = 0;

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

if (arr[i][j] != 0) { val++; }

}

if (val == 0) { udalenie(arr, i + 1); }

}

}

void BFSD(vector <vector <int>> gr, int v) {

queue <unsigned int> q = {};

q.push(v);

DIST[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

printf(" %d", v + 1);

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

if (gr[v][i] > 0 && DIST[i] > DIST[v] + gr[v][i]) {

q.push(i);

DIST[i] = DIST[v] + gr[v][i];

}

}

}

}

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

{

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

vector < vector <int> > m;

bool orient, vzves;

if (argc != 3) {

cout << "Количество аргументов больше или меньше двух" << endl;

return 0;

}

if (strcmp(argv[1], "-nor") != 0) {

if (strcmp(argv[1], "-or") != 0) {

cout << "Первый аргумент указан не верно" << endl;

return 0;

}

}

if (strcmp(argv[2], "-nvz") != 0) {

if (strcmp(argv[2], "-vz") != 0) {

cout << "Второй аргумент указан не верно" << endl;

return 0;

}

}

if (strcmp(argv[1], "-or") == 0) orient = 1;

else orient = 0;

if (strcmp(argv[2], "-vz") == 0) vzves = 1;

else vzves = 0;

int n;

printf("Введите кол-во вершин: ");

cin >> n;

m.resize(n);

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

m.at(i).resize(n);

}

srand(time(NULL));

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

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

if (i == j) {

m[i][j] = 0;

if (!orient) break;

}

else {

int z = rand() % 101;

if (!vzves) {

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = 1; }

}

else {

if (z < 30) { m[i][j] = 0; }

else { m[i][j] = z; }

}

if (!orient) m[j][i] = m[i][j];

}

}

}

clear(m);

if (n > m.size()) printf("\nВершины без рёбер удалены.\n");

printf("\nМатрица смежности графа\n\n");

vivod\_m(m);

int v;

printf("\nВведите вершину, для которой нужно найти расстояния: ");

cin >> v;

DIST.resize(n);

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

DIST[i] = INT\_MAX;

}

printf("\nПройденный путь:");

BFSD(m, v - 1);

printf("\n");

printf("\nРасстояния от точки %d до других точек: \n", v);

printf("\n|");

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

printf("%-4d|", i + 1);

}

printf("\n|");

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

printf("%-4d|", DIST[i]);

}

printf("\n");

system("pause");

return 0;

}

**Результаты:**

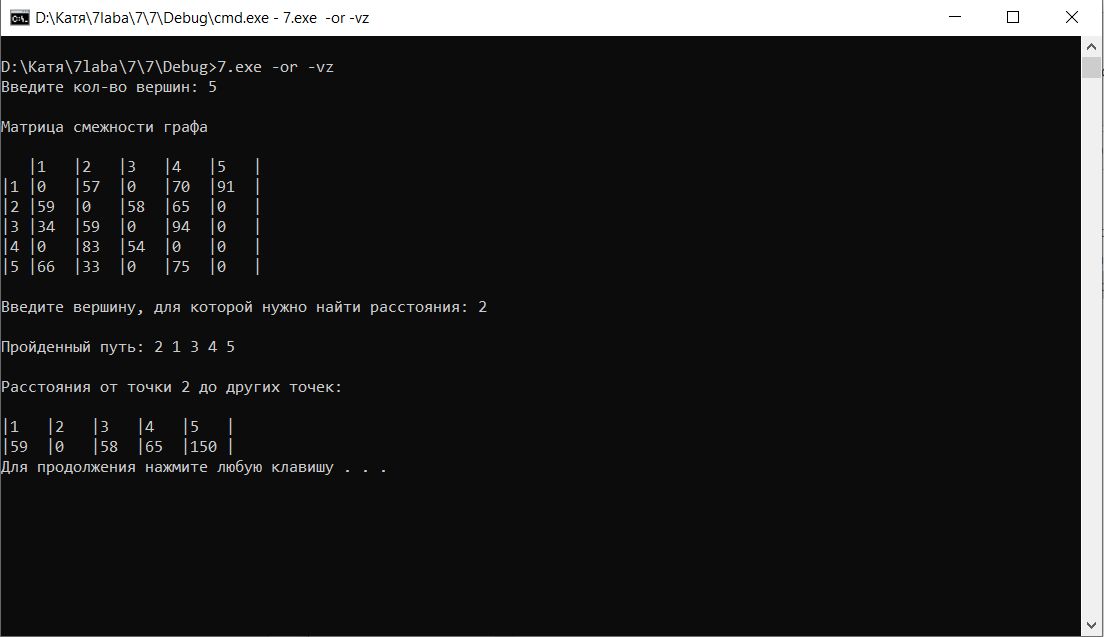


Рисунок 1 – Ориентированный взвешенный граф

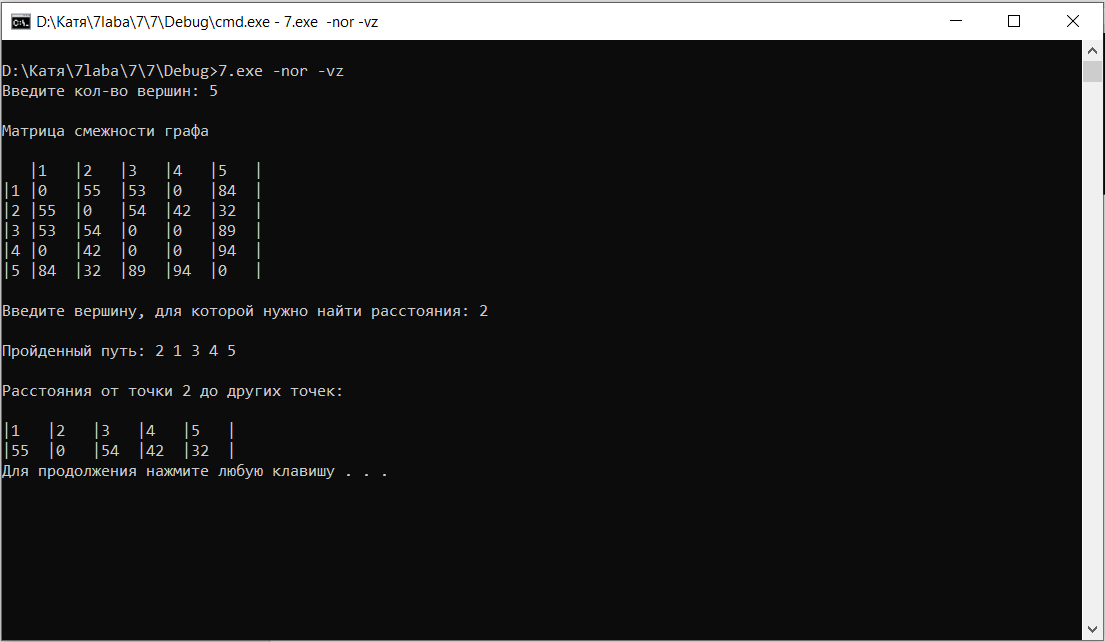


Рисунок 2 – Неориентированный взвешенный граф

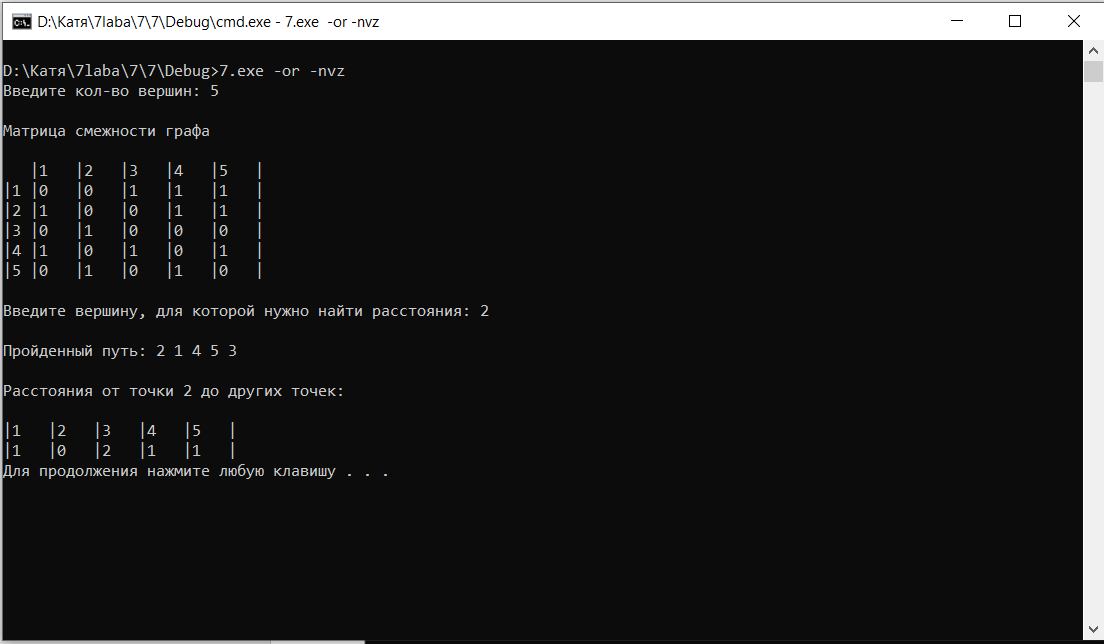


Рисунок 3 – Ориентированный невзвешенный граф

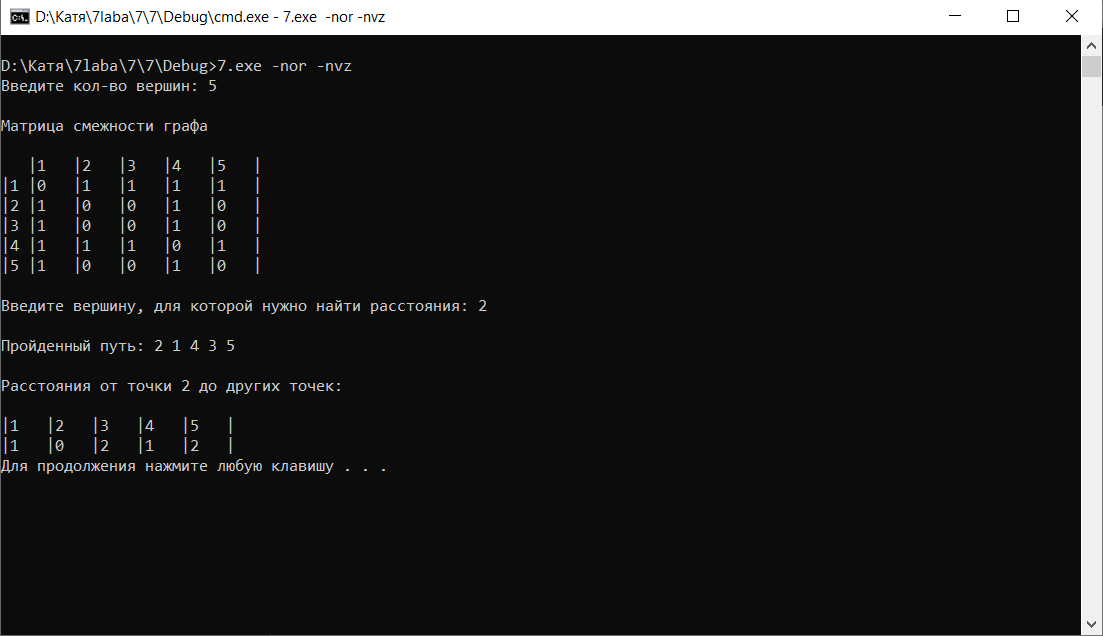


Рисунок 4 – Неориентированный невзвешенный граф