Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №10

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний во взвешенном графе»

**Выполнили студенты группы 21вв1:**

Кирьянов В.Е.

Аляев А.О.

**Приняли**

Юрова О.В.

Акифьев И.В.

Пенза 2022

**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. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

**Задание 3\***

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).

**Листинг программы:**

**Задание 1:**

#include "StdAfx.h"

#include <iostream>

#include <queue>

#include <stack>

#include "time.h"

using namespace std;

int\*\* G;

int\* dist;

int radius=INT\_MAX;

int dia = 0;

int BFSD(int size,int v)

{

dist = new int [size];

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

dist[i] = -1;

//const int size = 7;

queue<int> Queue;

Queue.push(v);

dist[v]=0;

while (!Queue.empty())

{

int v = Queue.front();

Queue.pop();

//cout << v << " ";

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

{

if (G[v][i] != 0 && dist[i] == -1)

{ // если вершина смежная и не обнаружена

Queue.push(i);

dist[i] = dist[v]+G[v][i];

}

}

// cout << node + 1 << " ";

}

return 0;

}

int main(){

setlocale(LC\_ALL, "Russian");

int size;

cout << "Введите размер матрицы: ";

cin >> size;

G = (int\*\*)malloc(sizeof(int\*) \* size);

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

G[i] = (int\*)malloc(sizeof(int) \* size);

}

srand(time(NULL));

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

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

G[i][j] = rand() % 2;

}

}

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

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

if (i == j) {

G[i][j] = 0;

}

if (G[i][j] = G[j][i]) {

G[i][j] = G[j][i];

}

}

}

for (size\_t i = 0; i < size; ++i) {

for (size\_t j = 0; j < size; ++j){

printf("%3d", G[i][j]);

}

printf("\n");

}

cout << "Расстояния в графе: ";

vector<int> Ekstrin;

int Eks\_max=0;

int vEks\_max=0;

int vEks\_min=10e4;

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

BFSD(size, i);

cout <<"\nРасстояние от "<< i <<" вершины: ";

cout << "\n";

Eks\_max=0;

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

cout <<"\nДо "<< j <<" вершины " << dist[j] << "";

if(dist[j]>Eks\_max)

Eks\_max=dist[j];

}

Ekstrin.push\_back(Eks\_max);

cout <<"\nМаксимальное расстояние: "<< Eks\_max << "\n";

cout << "\n";

}

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

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

if(Ekstrin[j]>vEks\_max)

vEks\_max=Ekstrin[j];

if(Ekstrin[j]<vEks\_min)

vEks\_min=Ekstrin[j];

}

}

cout << "\n";

cout << "Радиус графа: "<< vEks\_min<<"\n";

cout << "Диаметр графа: "<< vEks\_max<<"\n";

cout << "\nПереферийные вершины графа: ";

int temp = Ekstrin.size();

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

// cout << Ekstrin[i] << " ";

if(Ekstrin[i]==vEks\_max)

cout << i << " ";

}

cout << "\nЦентральные вершины графа: ";

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

// cout << Ekstrin[i] << " ";

if(Ekstrin[i]==vEks\_min)

cout << i << " ";

}

cout << "\n";

}

**Задание 2:**

#include "StdAfx.h"

#include <iostream>

#include <queue>

#include <stack>

#include "time.h"

using namespace std;

int\*\* G;

int radius=INT\_MAX;

int diametr = 0;

int\* dist;

int BFSD(int size,int v)

{

dist = new int [size];

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

dist[i] = -1;

//const int size = 7;

queue<int> Queue;

Queue.push(v);

dist[v]=0;

while (!Queue.empty())

{

int v = Queue.front();

Queue.pop();

//cout << v << " ";

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

{

if (G[v][i] != 0 && dist[i] == -1)

{ // если вершина смежная и не обнаружена

Queue.push(i);

dist[i] = dist[v]+G[v][i];

}

}

// cout << node + 1 << " ";

}

return 0;

}

int main(){

setlocale(LC\_ALL, "Russian");

int size;

cout << "Введите размер матрицы: ";

cin >> size;

G = (int\*\*)malloc(sizeof(int\*) \* size);

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

G[i] = (int\*)malloc(sizeof(int) \* size);

}

srand(time(NULL));

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

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

G[i][j] = rand() % 2;

if(G[i][j] == 1)

G[i][j] = rand()%100 +1;

else

G[i][j] = 0;

}

}

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

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

if (i == j) {

G[i][j] = 0;

}

if ((G[i][j] > 0)&&(G[j][i] > 0)) {

G[j][i] = 0;

}

}

}

for (size\_t i = 0; i < size; ++i) {

for (size\_t j = 0; j < size; ++j){

printf("%3d", G[i][j]);

}

printf("\n");

}

cout << "Расстояния в графе: ";

stack<int> Centr;

stack<int> Diametr;

Diametr.push(10);

Centr.push(10);

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

BFSD(size, i);

cout <<"\nРасстояние от "<< i <<" вершины: ";

cout << "\n";

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

cout <<"\nДо "<< j <<" вершины " << dist[j] << "";

if(diametr<=dist[j]&&(dist[j]>0)){

diametr=dist[j];

Diametr.pop();

Diametr.push(i);

}

if((radius>=dist[j])&&(dist[j]>0)){

radius=dist[j];

Centr.pop();

Centr.push(i);

}

}

cout << "\n";

}

cout << "\n";

cout << "Радиус графа: "<< radius<<"\n";

cout << "Диаметр графа: "<< diametr<<"\n";

cout << "\nПереферийные вершины графа: ";

while(!Diametr.empty()){

int temp = Diametr.top();

Diametr.pop();

cout << temp << " ";

}

cout << "\nЦентральные вершины графа: ";

while(!Centr.empty()){

int temp = Centr.top();

Centr.pop();

cout << temp << " ";

}

}

**Задание 3:**

// Lab\_10\_task3.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <cstring>

#include <queue>

#include <stack>

#include "time.h"

using namespace std;

int\*\* G;

int One,Two;

int size;

int\* dist;

int radius=10e5;

int diametr = 0;

bool orien = false;

bool vzvesh = false;

int BFSD(int size,int v)

{

dist = new int [size];

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

dist[i] = -1;

//const int size = 7;

queue<int> Queue;

Queue.push(v);

dist[v]=0;

while (!Queue.empty())

{

int v = Queue.front();

Queue.pop();

//cout << v << " ";

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

{

if (G[v][i] != 0 && dist[i] == -1)

{ // если вершина смежная и не обнаружена

Queue.push(i);

dist[i] = dist[v]+G[v][i];

}

}

// cout << node + 1 << " ";

}

return 0;

}

int\*\* generate(int size, int\*\* G, bool vzvesh, bool orien){

srand(time(NULL));

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

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

G[i][j] = rand() % 2;

if((G[i][j] == 1)&&(vzvesh==true))

G[i][j] = rand()%100 +1;

else if(vzvesh==false)

G[i][j] = rand() % 2;

}

}

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

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

if (i == j) {

G[i][j] = 0;

}

if(orien == true){

if ((G[i][j] > 0)&&(G[j][i] > 0)) {

G[j][i] = 0;

}

}

else if(orien == false){

if (G[i][j] = G[j][i]) {

G[i][j] = G[j][i];

}

}

}

}

for (size\_t i = 0; i < size; ++i) {

for (size\_t j = 0; j < size; ++j){

printf("%3d", G[i][j]);

}

printf("\n");

}

return G;

}

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

{

setlocale(LC\_CTYPE, "rus");

cout << "Введите размер матрицы: ";

cin >> size;

G = (int\*\*)malloc(sizeof(int\*) \* size);

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

G[i] = (int\*)malloc(sizeof(int) \* size);

}

if(argc > 1)

{

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

cout << "Взвешенный граф\n";

vzvesh = true;

}

else if(strcmp(argv[1], "0") == 0){

cout << "Невзвешенный граф\n";

vzvesh = false;

}

if(strcmp(argv[2], "1") == 0){

cout << "Ориентированный граф\n\n";

orien = true;

}

else if(strcmp(argv[2], "0") == 0){

cout << "Неориентированный граф\n\n";

orien = false;

}

generate(size,G,vzvesh,orien);

}

else

{

cout << "Укажите аргументы. ";

}

cout << "Расстояния в графе: ";

stack<int> Centr;

stack<int> Diametr;

Diametr.push(10);

Centr.push(10);

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

BFSD(size, i);

cout <<"\nРасстояние от "<< i <<" вершины: ";

cout << "\n";

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

cout <<"\nДо "<< j <<" вершины " << dist[j] << "";

if(diametr<=dist[j]&&(dist[j]>0)){

diametr=dist[j];

Diametr.pop();

Diametr.push(i);

}

if((radius>=dist[j])&&(dist[j]>0)){

radius=dist[j];

Centr.pop();

Centr.push(i);

}

}

cout << "\n";

}

cout << "\n";

cout << "Радиус графа: "<< radius<<"\n";

cout << "Диаметр графа: "<< diametr<<"\n";

cout << "\nПереферийные вершины графа: ";

while(!Diametr.empty()){

int temp = Diametr.top();

Diametr.pop();

cout << temp << " ";

}

cout << "\nЦентральные вершины графа: ";

while(!Centr.empty()){

int temp = Centr.top();

Centr.pop();

cout << temp << " ";

}

cout << '\n';

getchar();

return 0;

}

**Ход работы:**

Результат работы программы поиска расстояний, радиуса и диаметра графа, а также центральных и периферийных вершин в невзвешенном, неориентированном графе:

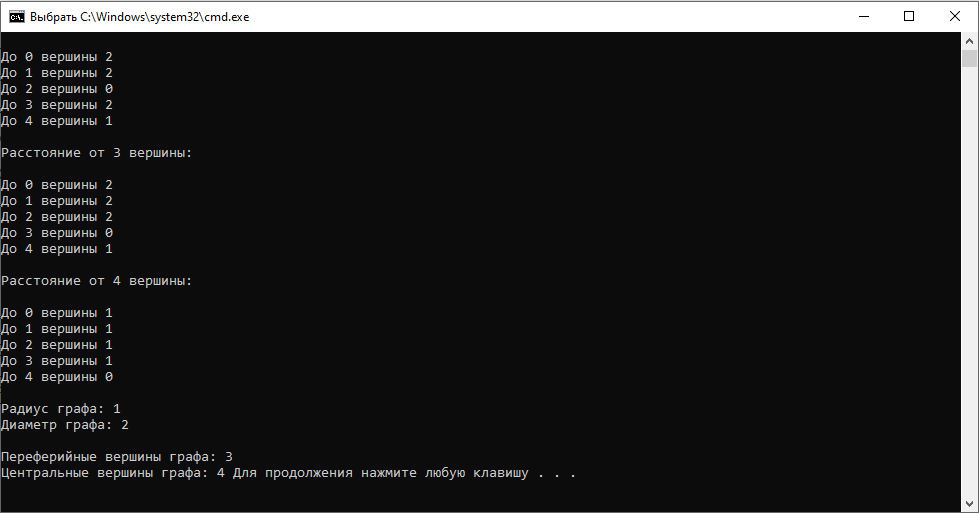
****

Рисунок 1 – результат работы программы.

Результат работы программы поиска расстояний, радиуса и диаметра графа, а также центральных и периферийных вершин в взвешенном, ориентированном графе:

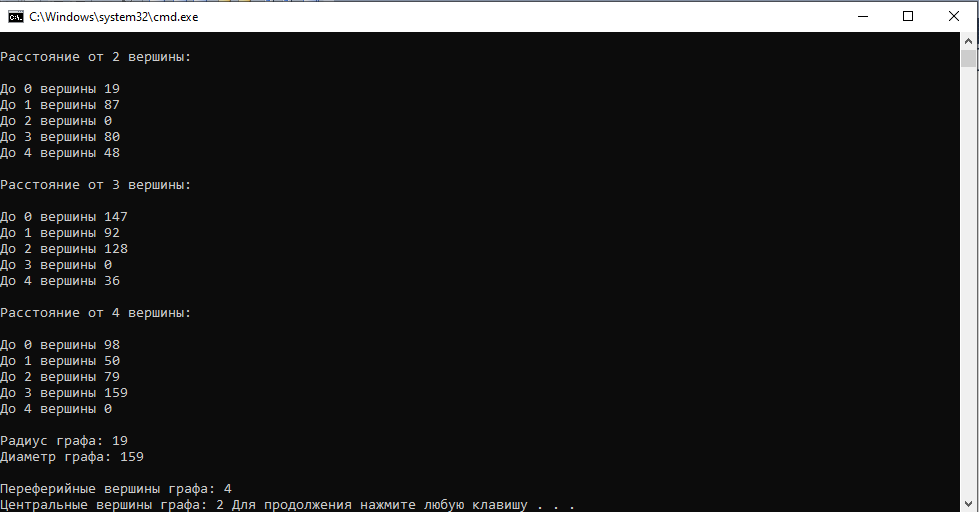


Рисунок 2 – результат работы программы.

**Компиляция программы через командную строку:**

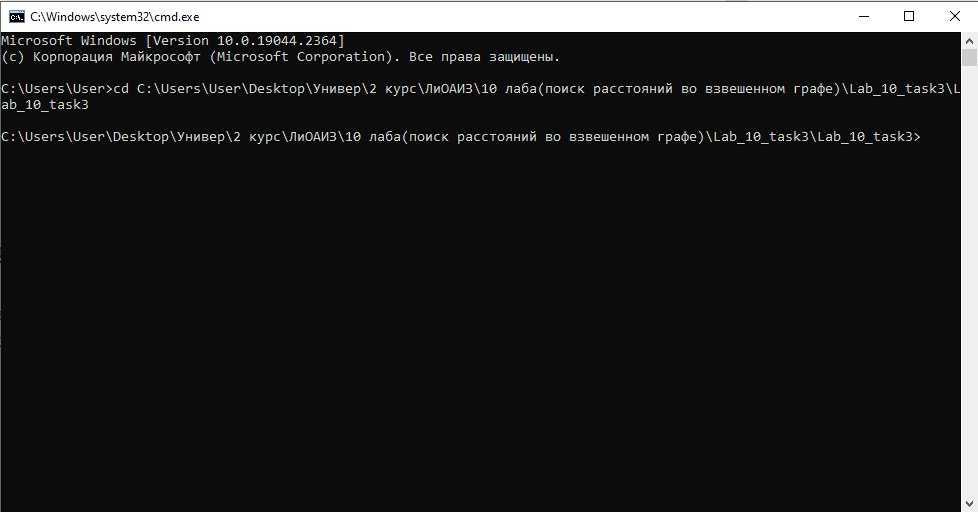
Перешли в каталог с .cpp файлом программы:

Рисунок 3 – работа с командной строкой.

Скомпилировали исполнительный .exe файл:

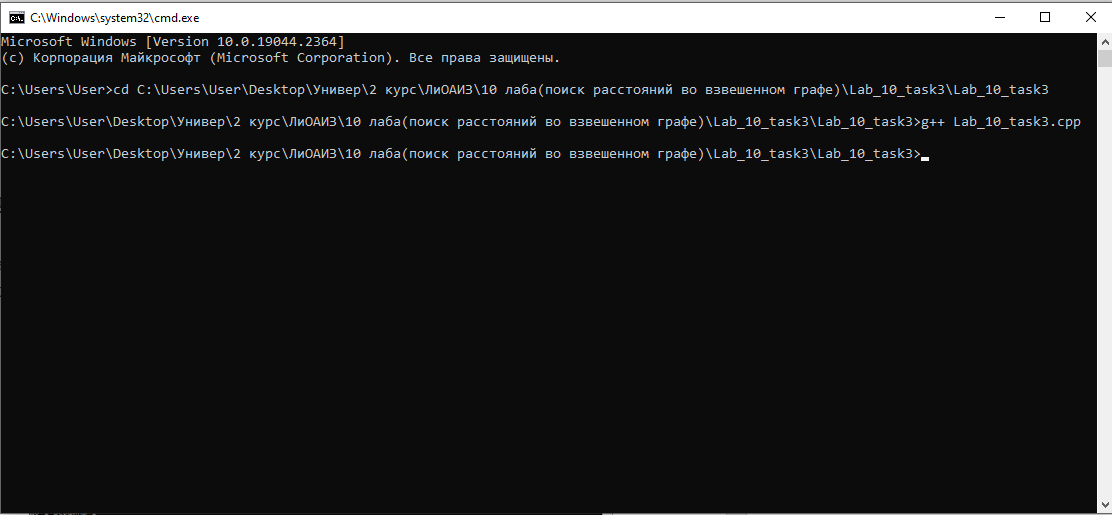
****

Рисунок 4 – компиляция исполнительного файла.

Вызвали исполнительный файл a.exe с параметрами взвешенного неориентированного графа:

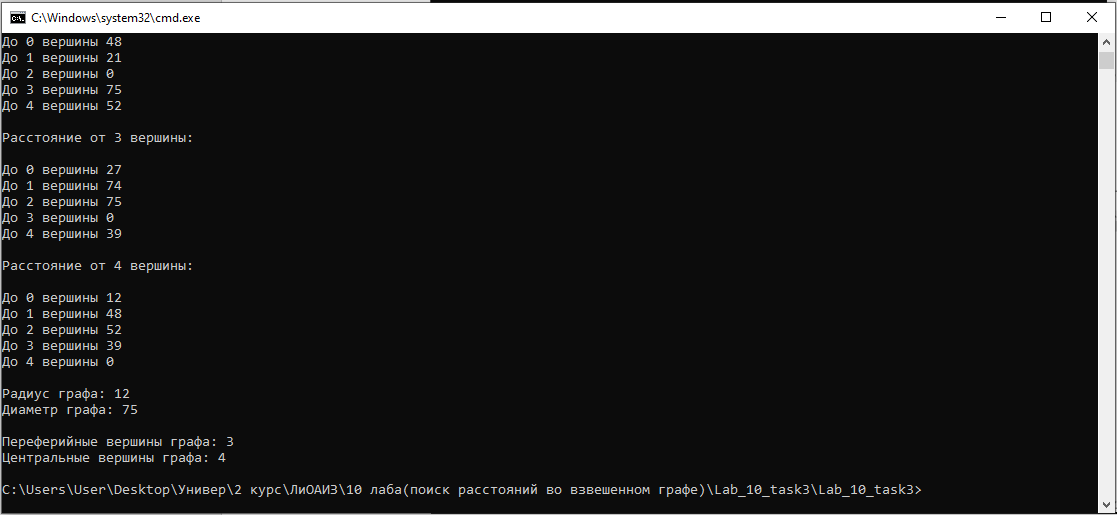
****

Рисунок 5 – результат работы программы.

**Вывод:** в ходе выполнения лабораторной работы были разработана программа, выполняющая поиск расстояний, радиуса и диаметра графа, а также центральных и периферийных вершин в различных видах графов.

Получили опыт в создании проектов в среде Microsoft Visual Studio, научились писать и отлаживать программы с применением поиска в ширину на языке Си. Научились создавать и запускать исполнительные .exe файлы из командной строки.