Министерство науки и высшего 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 22ВВВ1

Дунюшкин В.А.

Ивлюшин М.Д.

Проверили:

к.т.н, доцентЮрова О.В.

к.э.н, доцентАкифьев И.В.

Пенза 2023

**Цель работы:**

Научиться проводить расчет расстояния во взвешенном или ориентированном графе. Научиться искать радиус графа, диаметр, центр и центр тяжести графа.

**Лабораторное задание:**

**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**

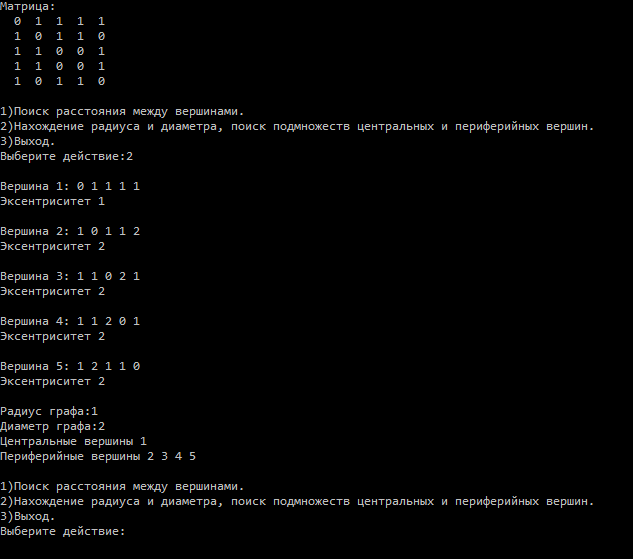
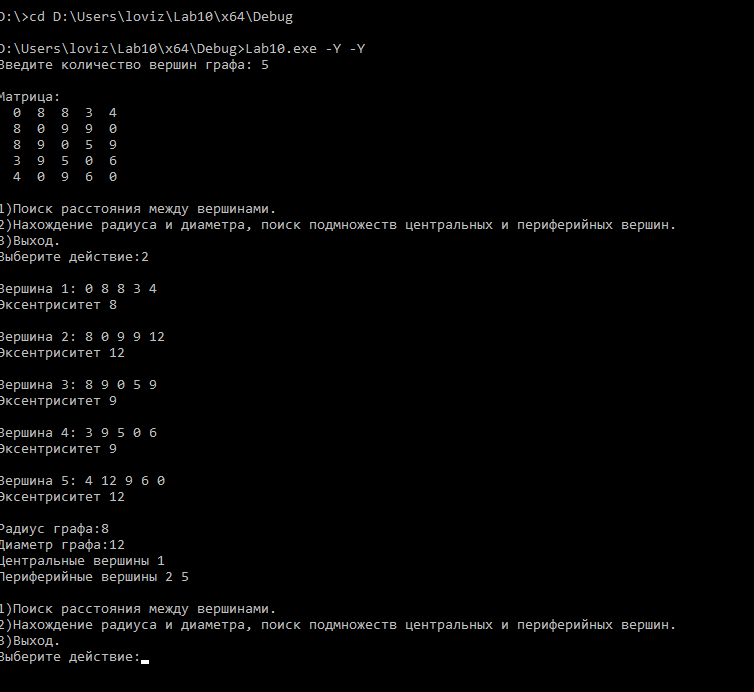
1. Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.
2. Определите подмножества периферийных и центральных вершин.

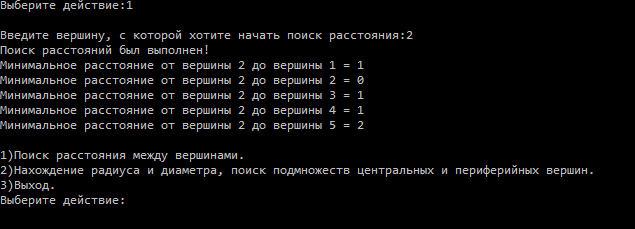
**Задание 3\***

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).

.

**Результаты работы программы:**

****

****

**Вывод**

В процессе выполнения лабораторной работы был освоен алгоритм поиска кратчайшего пути в графе с весами на рёбрах. Написанный программный код позволяет создавать взвешенные графы, хранить их в виде матрицы смежности или списков смежности, а также выполнять поиск кратчайших путей с помощью алгоритма поиска в ширину. Этот опыт позволил лучше разобраться в работе с взвешенными графами, а также в их особенностях и применении для вычисления кратчайших путей в различных задачах.

**Листинг**

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <queue>

#include <iomanip>

using namespace std;

int i, j;

typedef struct GraphSettings {

int weighted; // 1, если граф взвешенный, иначе 0

int orientation; // 1, если граф ориентированный, иначе 0

} GS;

GS values(int argc, char\* argv[])

{

GS val = { 0,0 };

for (i = 1; i < argc; i++)

{

if (strcmp(argv[i], "-Y") == 0)

{

val.weighted = 1;

}

else if (strcmp(argv[i], "-Y") == 0)

{

val.orientation = 1;

}

}

return val;

}

void SearchDist(int\*\* smej, int ver, int ver1, int\* dist)

{

int newdist = 0;

queue<int> q; // пустая очередь

dist[ver1 - 1] = 0;

q.push(ver1);

while (!q.empty())

{

int current = q.front();

q.pop();

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

{

if (smej[current - 1][i] > 0)

{

newdist = dist[current - 1] + smej[current - 1][i];

if (newdist < dist[i])

{

dist[i] = newdist;

q.push(i + 1);

}

}

}

}

}

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

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

GS gs = values(argc, argv);

int\*\* smej = 0;

int\* visit = 0;

int\* dist = 0;

int\* eksen = 0;

int choice, ver, ver1, k = 0, first = 0, MaxDist = 0, D = 0, R = 0;

bool prov1 = true;

cout << "Введите количество вершин графа: ";

for (;;) {

cin >> ver;

if (cin.fail() || ver < 0) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный номер количество вершин графа! Повторите попытку: ";

}

else { break; }

}

cout << endl;

//Выделение памяти под массив смежности

smej = new int\* [ver];

if (smej == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

smej[i] = new int[ver];

}

//Генерация массива смежности

for (i = 0; i < ver; i++)

{

for (j = i; j < ver; j++)

{

if (i == j)

{

smej[i][j] = 0; // на главной диагонали нули

}

else

{

if (gs.weighted == 1 && gs.orientation == 1)

{

smej[i][j] = (-10) + rand() % 19;

if (smej[i][j] < 0)

{

smej[i][j] = 0;

smej[j][i] = rand() % 10;

}

else smej[j][i] = 0;

}

else if (gs.weighted == 0 && gs.orientation == 1)

{

smej[i][j] = (-1) + rand() % 3;

if (smej[i][j] < 0)

{

smej[i][j] = 0;

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

}

else smej[j][i] = 0;

}

else if (gs.weighted == 1 && gs.orientation == 0)

{

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

if (smej[i][j] = 1) smej[i][j] = rand() % 10;

smej[j][i] = smej[i][j];

}

else if (gs.weighted == 0 && gs.orientation == 0)

{

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

smej[j][i] = smej[i][j];

}

}

}

}

//Вывод матрицы смежности

cout << "Матрица:" << endl;

for (i = 0; i < ver; i++)

{

for (j = 0; j < ver; j++)

{

cout << setw(3) << smej[i][j];

}

cout << endl;

}

do

{

cout << endl;

cout << "1)Поиск расстояния между вершинами." << endl;

cout << "2)Нахождение радиуса и диаметра, поиск подмножеств центральных и периферийных вершин." << endl;

cout << "3)Выход." << endl;

cout << "Выберите действие:";

cin >> choice;

switch (choice)

{

case 1:

dist = new int[ver];

if (dist == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

dist[i] = INT\_MAX;

}

do

{

cout << endl;

cout << "Введите вершину, с которой хотите начать поиск расстояния:";

cin >> ver1;

if (cin.fail() || ver1 < 1 || ver1 > ver)

{

cout << "Некорректный номер вершины! Повторите попытку." << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else

{

for (i = 0; i < ver; i++)

{

if (smej[ver1 - 1][i] != 0 || smej[i][ver1 - 1] != 0)

{

k++;

}

}

if (k == 0)

cout << "Вершина " << ver1 << " изолированная! Введите другую начальную вершину." << endl;

else

prov1 = false;

}

} while (prov1);

cout << "Поиск расстояний был выполнен!" << endl;

SearchDist(smej, ver, ver1, dist);

for (i = 0; i < ver; i++)

{

if (dist[i] == INT\_MAX)

cout << "Вершина " << ver1 << " не имеет пути в вершину " << i + 1 << endl;

else

cout << "Минимальное расстояние от вершины " << ver1 << " до вершины " << i + 1 << " = " << dist[i] << endl;

}

delete[] dist;

break;

case 2:

D = INT\_MIN;

R = INT\_MAX;

visit = new int[ver];

if (visit == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

eksen = new int[ver];

if (eksen == NULL)

{

cout << "Не удалось выделить память!" << endl;

return;

}

for (i = 0; i < ver; i++)

{

eksen[i] = 0;

}

for (first = 0; first < ver; first++)

{

for (i = 0; i < ver; i++)

{

visit[i] = INT\_MAX;

}

SearchDist(smej, ver, first + 1, visit);

MaxDist = 0;

for (i = 0; i < ver; i++)

{

if (visit[i] > MaxDist && visit[i] < 2700000)

{

MaxDist = visit[i];

}

else continue;

}

cout << endl << "Вершина " << first + 1 << ": ";

for (int firs = 0; firs < ver; firs++)

{

if(visit[firs]< 2700000)

cout << visit[firs] << " ";

else

cout << 0 << " ";

}

cout << endl;

eksen[first] = MaxDist;

cout << "Эксентриситет " << eksen[first] << endl;

}

for (i = 0; i < ver; i++)

{

if (eksen[i] > D)

{

D = eksen[i];

}

}

R = D;

for (i = 0; i < ver; i++)

{

if (eksen[i] < R)

{

R = eksen[i];

}

}

cout << endl << "Радиус графа:" << R << endl;

cout << "Диаметр графа:" << D << endl;

cout << "Центральные вершины";

for (i = 0; i < ver; i++)

{

if (eksen[i] == R)

{

cout << " " << i + 1;

}

}

cout << endl << "Периферийные вершины";

for (i = 0; i < ver; i++)

{

if (eksen[i] == D)

{

cout << " " << i + 1;

}

}

cout << endl;

delete[] visit;

delete[] eksen;

break;

case 3:

system("cls");

break;

default:

cout << endl;

cout << "Неверный выбор! Повторите попытку.";

cout << endl;

break;

}

} while (choice != 3);

for (i = 0; i < ver; i++)

{

delete[] smej[i];

}

delete[] smej;

}