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 21ВВ3

Чинов Даниил

Тюкалов Василий

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2022

# Название

Поиск расстояний во взвешенном графе

**Цель работы** – изучение алгоритма поиск расстояний во взвешенном графе.

# 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).

# Листинг

#include <iostream>

#include <locale>

#include <queue>

#include <stack>

using namespace std;

int max(int\* arr, int count) {

int m = 0;

for (int i = 0; i != count; i++) {

if (m < arr[i] && arr[i] != 10000) m = arr[i];

}

return m;

}

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

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int count = 2;

cout << "Введите размер матрицы: ";

cin >> count;

clock\_t start, end; // объявляем переменные для определения времени выполнения

/\*

1. Сгенерируйте (используя генератор случайных чисел)

матрицу смежности для неориентированного графа G.

Выведите матрицу на экран.

\*/

int\*\* arr1 = new int\* [count];

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

arr1[i] = new int[count];

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

arr1[i][j] = 0;

}

}

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

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

int p = rand() % 101;

if (i == j) {

arr1[i][j] = 0;

continue;

}

if (p > 60) {

int weight = rand() % 10;

arr1[i][j] = weight;

arr1[j][i] = weight;

}

}

}

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

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

cout << arr1[i][j] << ' ';

}

cout << endl;

}

queue<int> Queue;

int\* visited = new int[count];

int\* dist = new int[count];

int temp, minindex, min;

int diam = 0;

int radi = 10000;

vector<int> max\_top; // Переферифные вершины

vector<int> min\_top; // Центральные вершины

cout << endl << endl;

for (int i = 0; i != count; i++) {

int begin\_index = i;

//Инициализация вершин и расстояний

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

{

dist[i] = 10000;

visited[i] = 1;

}

dist[begin\_index] = 0;

// Шаг алгоритма

do {

minindex = 10000;

min = 10000;

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

{ // Если вершину ещё не обошли и вес меньше min

if ((visited[i] == 1) && (dist[i] < min))

{ // Переприсваиваем значения

min = dist[i];

minindex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minindex != 10000)

{

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

{

if (arr1[minindex][i] > 0)

{

temp = min + arr1[minindex][i];

if (temp < dist[i])

{

dist[i] = temp;

}

}

}

visited[minindex] = 0;

}

} while (minindex < 10000);

// Вывод кратчайших расстояний до вершин

printf("\nКратчайшие расстояния до вершин: \n");

for (int i = 0; i < count; i++) printf("%-6d ", dist[i]);

int g = max(dist, count);

cout << " эксцентреситет = " << g << endl;

if (g < radi && g != 0) {

radi = g;

min\_top.clear();

min\_top.push\_back(i);

}

else if (g == radi) min\_top.push\_back(i);

if (g > diam) {

diam = g;

max\_top.clear();

max\_top.push\_back(i);

}

else if (g == diam) max\_top.push\_back(i);

}

if (diam > 0 && radi < 10000) {

cout << endl << endl << "Диаметр = " << diam << endl;

cout << "Радиус = " << radi << endl;

cout << "периферийные вершины: ";

for (int i = 0; i != max\_top.size(); i++) {

cout << max\_top[i] << ' ';

}

cout << endl << "Центральные вершины: ";

for (int i = 0; i != min\_top.size(); i++) {

cout << min\_top[i] << ' ';

}

cout << endl;

for (int i = 0; i != count; i++) {

int edge\_count = 0;

for (int j = 0; j != count; j++) {

if (arr1[i][j] != 0) edge\_count++;

}

if (edge\_count == 0) cout << "Вершина " << i << " Изолированная" << endl;

else if (edge\_count == 1) cout << "Вершина " << i << " Концевая" << endl;

else if (edge\_count == count - 1) cout << "Вершина " << i << " Доминирующая " << endl;

else cout << "Вершина " << i << " просто вершина" << endl;

}

}

else {

cout << endl << endl << "Диаметра и Радиуса нет в данном графе" << diam << endl;

cout << "периферийных вершин и Центральные вершин в данном графе нет" << endl << endl;

}

/\*

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.

\*/

cout << endl << endl;

int count2 = 2;

cout << "Введите размер матрицы: ";

cin >> count2;

int\*\* arr2 = new int\* [count2];

int\* visited2 = new int[count2];

int\* dist2 = new int[count2];

diam = 0;

radi = 10000;

min\_top.clear();

max\_top.clear();

for (int i = 0; i < count2; i++) {

arr2[i] = new int[count2];

for (int j = 0; j < count2; j++) {

arr2[i][j] = 0;

}

}

for (int i = 0; i < count2; i++) {

for (int j = 0; j < count2; j++) {

int p = rand() % 101;

if (i == j) {

arr2[i][j] = 0;

continue;

}

if (p > 60) {

int weight = rand() % 10;

arr2[i][j] = weight;

}

}

}

for (int i = 0; i < count2; i++) {

for (int j = 0; j < count2; j++) {

cout << arr2[i][j] << ' ';

}

cout << endl;

}

cout << endl << endl;

for (int i = 0; i != count2; i++) {

int begin\_index = i;

//Инициализация вершин и расстояний

for (int i = 0; i < count2; i++)

{

dist2[i] = 10000;

visited2[i] = 1;

}

dist2[begin\_index] = 0;

// Шаг алгоритма

do {

minindex = 10000;

min = 10000;

for (int i = 0; i < count2; i++)

{ // Если вершину ещё не обошли и вес меньше min

if ((visited2[i] == 1) && (dist2[i] < min))

{ // Переприсваиваем значения

min = dist2[i];

minindex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minindex != 10000)

{

for (int i = 0; i < count2; i++)

{

if (arr2[minindex][i] > 0)

{

temp = min + arr2[minindex][i];

if (temp < dist2[i])

{

dist2[i] = temp;

}

}

}

visited2[minindex] = 0;

}

} while (minindex < 10000);

// Вывод кратчайших расстояний до вершин

printf("\nКратчайшие расстояния до вершин: \n");

for (int i = 0; i < count2; i++) printf("%-6d ", dist2[i]);

int g = max(dist2, count2);

cout << " эксцентреситет = " << g << endl;

if (g < radi && g != 0) {

radi = g;

min\_top.clear();

min\_top.push\_back(i);

}

else if (g == radi) min\_top.push\_back(i);

if (g > diam) {

diam = g;

max\_top.clear();

max\_top.push\_back(i);

}

else if (g == diam) max\_top.push\_back(i);

}

if (diam > 0 && radi < 10000) {

cout << endl << endl << "Диаметр = " << diam << endl;

cout << "Радиус = " << radi << endl;

cout << "периферийные вершины: ";

for (int i = 0; i != max\_top.size(); i++) {

cout << max\_top[i] << ' ';

}

cout << endl << "Центральные вершины: ";

for (int i = 0; i != min\_top.size(); i++) {

cout << min\_top[i] << ' ';

}

cout << endl;

for (int i = 0; i != count2; i++) {

int edge\_count = 0;

for (int j = 0; j != count2; j++) {

if (arr2[i][j] != 0) edge\_count++;

}

if (edge\_count == 0) cout << "Вершина " << i << " Изолированная" << endl;

else if (edge\_count == 1) cout << "Вершина " << i << " Концевая" << endl;

else if (edge\_count == count2 - 1) cout << "Вершина " << i << " Доминирующая " << endl;

else cout << "Вершина " << i << " просто вершина" << endl;

}

}

else {

cout << endl << endl << "Диаметра и Радиуса нет в данном графе" << diam << endl;

cout << "периферийных вершин и Центральные вершин в данном графе нет" << endl << endl;

}

/\*

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).

\*/

cout << endl << endl << endl;

if (argc != 3) {

cout << "конец программы" << endl;

return 0;

}

string a1 = argv[1];

string a2 = argv[2];

if ((a1 != "1" && a1 != "0") || (a2 != "1" && a2 != "0")) {

cout << "Введенны неправильные данные" << endl;

}

else {

int count3;

cout << "Введите размер графа: ";

cin >> count3;

int\*\* arr3;

arr3 = new int\* [count3];

int\* visited3 = new int[count2];

int\* dist3 = new int[count2];

diam = 0;

radi = 10000;

min\_top.clear();

max\_top.clear();

for (int i = 0; i < count3; i++) {

arr3[i] = new int[count3];

for (int j = 0; j < count3; j++) {

arr3[i][j] = 0;

}

}

for (int i = 0; i < count3; i++) {

for (int j = 0; j < count3; j++) {

int p = rand() % 101;

if (i == j) {

arr3[i][j] = 0;

continue;

}

if (p > 60) {

int weight;

if (a1 == "1") weight = rand() % 10;

else weight = 1;

if (a2 == "1") {

arr3[i][j] = weight;

}

else {

arr3[i][j] = weight;

arr3[j][i] = weight;

}

}

}

}

for (int i = 0; i < count3; i++) {

for (int j = 0; j < count3; j++) {

cout << arr3[i][j] << ' ';

}

cout << endl;

}

for (int i = 0; i != count3; i++) {

int begin\_index = i;

//Инициализация вершин и расстояний

for (int i = 0; i < count3; i++)

{

dist3[i] = 10000;

visited3[i] = 1;

}

dist3[begin\_index] = 0;

// Шаг алгоритма

do {

minindex = 10000;

min = 10000;

for (int i = 0; i < count3; i++)

{ // Если вершину ещё не обошли и вес меньше min

if ((visited3[i] == 1) && (dist3[i] < min))

{ // Переприсваиваем значения

min = dist3[i];

minindex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minindex != 10000)

{

for (int i = 0; i < count3; i++)

{

if (arr3[minindex][i] > 0)

{

temp = min + arr3[minindex][i];

if (temp < dist3[i])

{

dist3[i] = temp;

}

}

}

visited3[minindex] = 0;

}

} while (minindex < 10000);

// Вывод кратчайших расстояний до вершин

printf("\nКратчайшие расстояния до вершин: \n");

for (int i = 0; i < count3; i++) printf("%-6d ", dist3[i]);

int g = max(dist3, count3);

cout << " эксцентреситет = " << g << endl;

if (g < radi && g != 0) {

radi = g;

min\_top.clear();

min\_top.push\_back(i);

}

else if (g == radi) min\_top.push\_back(i);

if (g > diam) {

diam = g;

max\_top.clear();

max\_top.push\_back(i);

}

else if (g == diam) max\_top.push\_back(i);

}

if (diam > 0 && radi < 10000) {

cout << endl << endl << "Диаметр = " << diam << endl;

cout << "Радиус = " << radi << endl;

cout << "периферийные вершины: ";

for (int i = 0; i != max\_top.size(); i++) {

cout << max\_top[i] << ' ';

}

cout << endl << "Центральные вершины: ";

for (int i = 0; i != min\_top.size(); i++) {

cout << min\_top[i] << ' ';

}

cout << endl;

for (int i = 0; i != count3; i++) {

int edge\_count = 0;

for (int j = 0; j != count3; j++) {

if (arr1[i][j] != 0) edge\_count++;

}

if (edge\_count == 0) cout << "Вершина " << i << " Изолированная" << endl;

else if (edge\_count == 1) cout << "Вершина " << i << " Концевая" << endl;

else if (edge\_count == count3 - 1) cout << "Вершина " << i << " Доминирующая " << endl;

else cout << "Вершина " << i << " просто вершина" << endl;

}

}

else {

cout << endl << endl << "Диаметра и Радиуса нет в данном графе" << diam << endl;

cout << "периферийных вершин и Центральные вершин в данном графе нет" << endl << endl;

}

for (int i = 0; i != count3; i++) delete[] arr3[i];

delete[] arr3;

delete[] visited3;

delete[] dist3;

}

for (int i = 0; i != count; i++) delete[] arr1[i];

for (int i = 0; i != count2; i++) delete[] arr2[i];

delete[] arr1;

delete[] arr2;

delete[] visited;

delete[] visited2;

delete[] dist;

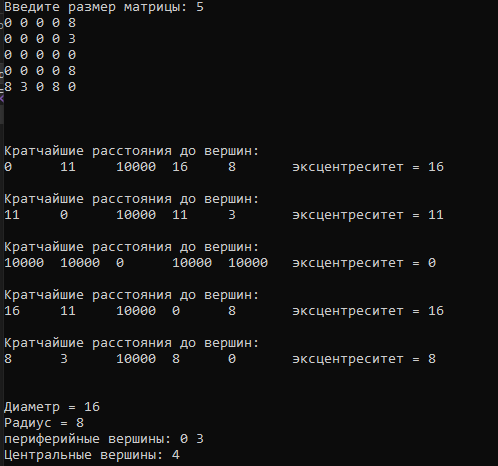
delete[] dist2;

return 0;

}

# Результат работы программы

Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

# Вывод

# В ходе выполнения лабораторной работы была разработана программа, в который был реализован алгоритм поиск расстояний во взвешенном графе.