Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2021

**ОТЧЁТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Определение характеристик графов»

Выполнили:

студентки группы 20ВВ4

Кожевникова А. В.

Мамелина Ю. В.

Приняла:

д.т.н Юрова О. В.

Пенза 2021

**Цель работы:** Изучить тему определения характеристик графов, а также ознакомится с понятиями, используемыми в лабораторной работе для выполнения практических заданий. Составить собственную программу в соответствии с методическими указаниями.

**Задание 1**

**1.** Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного взвешенного графа G. Выведите матрицу

на экран.

**2.** Определите радиус и диаметр графа G, используя матрицу смежности

графа.

**3.** Определите подмножества периферийных и центральных вершин

графа G, используя матрицу смежности.

**4.** Найдите изолированные, концевые и доминирующие вершины.

**Задание 2**\*

**1.** Постройте для графа G матрицу инцидентности.

**2.** Определите радиус и диаметр графа G, используя матрицу

инцидентности графа.

**3.** Определите подмножества периферийных и центральных вершин графа G, используя матрицу инцидентности.

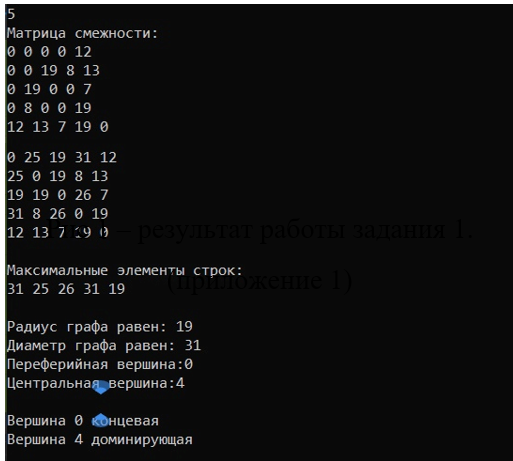


Рис 1 – результат работы задания 1.

(приложение 1)

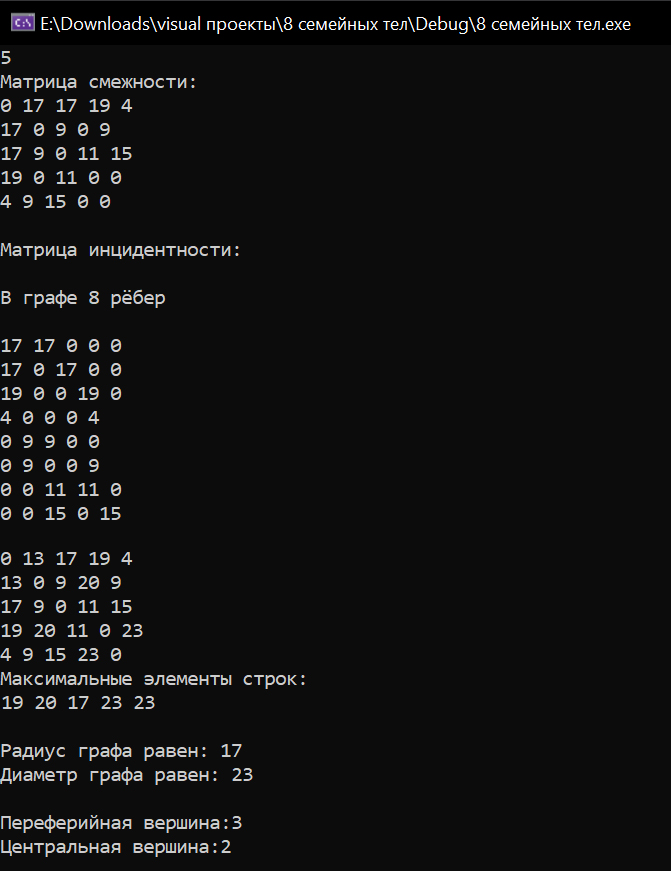


Рис 2 – результат работы задания 2.

(приложение 2)

**Вывод:** Изучили тему определения характеристик графов, а также ознакомились с понятиями, используемыми в лабораторной работе для выполнения практической части. Составили алгоритм решения поставленных задач.

**Приложение:**

Приложение 1

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

vector<int> BFSD\_for\_weighted\_matrix(int s, int\* m[], int n);

int main()

{

setlocale(LC\_ALL, "rus");

int n;

cin >> n;

int\*\* matrix = new int\* [n];

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

{

matrix[i] = new int[n];

}

srand(time(NULL));

cout << "Матрица смежности:" << endl;

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

{

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

{

matrix[i][j] = rand() % 101;

if (i == j) {

matrix[i][j] = 0;

}

if (matrix[i][j] >= 0 && matrix[i][j] <= 30) {

matrix[i][j] = 0;

}

if (matrix[i][j] >= 31 && matrix[i][j] <= 100) {

matrix[i][j] = rand() % 20 + 1;

}

}

}

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

{

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

{

matrix[i][j] = matrix[j][i];

cout << matrix[i][j] << " ";

}

cout << endl;

}

int v;

vector<vector<int>> dist\_mat;

dist\_mat.resize(n);

vector<int> mat(n);

//считываем расстояние для каждой вершины

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

{

v = i;

mat = BFSD\_for\_weighted\_matrix(v, matrix, n);

dist\_mat.push\_back(mat);

}

cout << endl;

//вывод матрицы расстояний

for (int i = 0; i < dist\_mat.size(); i++)

{

for (int j = 0; j < dist\_mat[i].size(); j++)

{

cout << dist\_mat[i][j] << " ";

}

cout << endl;

}

cout << endl;

//считываем максимальный элемент каждой строки матрицы

vector<int> max\_in\_row;

max\_in\_row.resize(n);

int max = -1;

cout << "Максимальные элементы строк:" << endl;

for (int i = 0; i < dist\_mat.size(); i++)

{

max = -1;

for (int j = 0; j < dist\_mat[i].size(); j++)

{

if (dist\_mat[i][j] >= max) {

max = dist\_mat[i][j];

}

}

max\_in\_row.push\_back(max);

}

for (int i = 0; i < max\_in\_row.size(); i++)

{

if (max\_in\_row[i] == -1 || max\_in\_row[i] == 0)

{

max\_in\_row.erase(max\_in\_row.begin() + 1);

}

cout << max\_in\_row[i] << " ";

}

cout << endl;

////Поиск диаметра и радиуса графа////

cout << endl;

auto diametre = max\_element(max\_in\_row.begin(), max\_in\_row.end());

auto radius = min\_element(max\_in\_row.begin() + n, max\_in\_row.end());

cout << "Радиус графа равен: " << \*radius << endl;

cout << "Диаметр графа равен: " << \*diametre << endl;

//////Поиск переферийных и центральных вершин//////

auto it = find(max\_in\_row.begin(), max\_in\_row.end(), \*diametre);

int index\_of\_per = it - max\_in\_row.begin();

cout << "Переферийная вершина:" << index\_of\_per - n << endl;

auto it\_for = find(max\_in\_row.begin(), max\_in\_row.end(), \*radius);

int index\_of\_cent = it\_for - max\_in\_row.begin();

cout << "Центральная вершина:" << index\_of\_cent - n << endl;

/////Поиск изолированных, концевых и доминирующие вершин////////

int\* collection\_of\_inc\_vertex = new int[n];

int count = 0;

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

{

count = 0;

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

{

if (matrix[i][j] > 0)

{

count++;

}

}

collection\_of\_inc\_vertex[i] = count;

}

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

{

if (collection\_of\_inc\_vertex[i] == 1)

{

cout << "Вершина " << i << " концевая" << endl;

}

if (collection\_of\_inc\_vertex[i] == 0)

{

cout << "Вершина " << i << " изолированная" << endl;

}

if (collection\_of\_inc\_vertex[i] == \*max\_element(collection\_of\_inc\_vertex, collection\_of\_inc\_vertex + n))

{

cout << "Вершина " << i << " доминирующая" << endl;

}

}

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

{

delete[] matrix[i];

}

delete[] matrix;

delete[] collection\_of\_inc\_vertex;

system("pause>nul");

return 0;

}

vector<int> BFSD\_for\_weighted\_matrix(int s, int\* m[], int n) {

vector<int> dist(n, INT\_MAX);

queue<int> q;

q.push(s);

dist[s] = 0;

int vis;

while (!q.empty()) {

vis = q.front();

q.pop();

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

if (m[vis][i] > 0 && dist[i] > dist[vis] + m[vis][i]) {

q.push(i);

dist[i] = dist[vis] + m[vis][i];

}

}

}

return dist;

}

Приложение 2.

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

//int\*\* AdjMatToIncmat(int\*\* AdjMatrix, int n);

vector<int> BFSD\_for\_weighted\_matrix(int s, int\* m[], int n, int edges);

int main()

{

setlocale(LC\_ALL, "rus");

int n;

cin >> n;

int\*\* matrix = new int\* [n];

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

{

matrix[i] = new int[n];

}

srand(time(NULL));

cout << "Матрица смежности:" << endl;

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

{

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

{

matrix[i][j] = rand() % 101;

if (i == j) {

matrix[i][j] = 0;

}

if (matrix[i][j] >= 0 && matrix[i][j] <= 30) {

matrix[i][j] = 0;

}

if (matrix[i][j] >= 31 && matrix[i][j] <= 100) {

matrix[i][j] = rand() % 20 + 1;

}

}

}

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

{

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

{

matrix[i][j] = matrix[j][i];

cout << matrix[i][j] << " ";

}

cout << endl;

}

cout << endl;

cout << "Матрица инцидентности:" << endl;

cout << endl;

int edges = 0;

//считаем количество рёбер в графе

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

{

for (int j = i + 1; j < n; j++)

{

if (matrix[i][j] > 0) {

edges++;

}

}

}

cout << "В графе " << edges << " рёбер" << endl;

cout << endl;

int\*\* incidence = new int\* [edges];

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

{

incidence[i] = new int[n];

}

int count = 0;

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

{

for (int j = i + 1; j < n; j++)

{

int edgNum = matrix[i][j];

if (edgNum > 0) {

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

{

if (k == i || k == j)

incidence[count][k] = matrix[i][j];

else

incidence[count][k] = 0;

}

count++;

}

}

}

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

{

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

{

cout << incidence[i][j] << " ";

}

cout << endl;

}

int v;

vector<vector<int>> dist\_mat;// (n, vector<int>(edges, 0));

dist\_mat.resize(n);

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

{

dist\_mat[i].resize(edges);

}

vector<int> mat(n);

//считываем расстояние для каждой вершины

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

{

v = i;

mat = BFSD\_for\_weighted\_matrix(v, matrix, n, edges);

dist\_mat.push\_back(mat);

}

cout << endl;

//вывод матрицы расстояний

for (int i = 0; i < dist\_mat.size(); i++)

{

for (int j = 0; j < dist\_mat[i].size(); j++)

{

cout << dist\_mat[i][j] << " ";

}

cout << endl;

}

vector<int> max\_in\_row;

max\_in\_row.resize(n);

int max = -1;

cout << "Максимальные элементы строк:" << endl;

for (int i = 0; i < dist\_mat.size(); i++)

{

max = -1;

for (int j = 0; j < dist\_mat[i].size(); j++)

{

if (dist\_mat[i][j] >= max) {

max = dist\_mat[i][j];

}

}

max\_in\_row.push\_back(max);

}

for (int i = 0; i < max\_in\_row.size(); i++)

{

if (max\_in\_row[i] == 0)

{

max\_in\_row.erase(max\_in\_row.begin() + 1);

}

cout << max\_in\_row[i] << " ";

}

cout << endl;

////Поиск диаметра и радиуса графа////

cout << endl;

auto diametre = max\_element(max\_in\_row.begin(), max\_in\_row.end());

auto radius = min\_element(max\_in\_row.begin() + n, max\_in\_row.end());

cout << "Радиус графа равен: " << \*radius << endl;

cout << "Диаметр графа равен: " << \*diametre << endl;

cout << endl;

//////Поиск переферийных и центральных вершин//////

auto it = find(max\_in\_row.begin(), max\_in\_row.end(), \*diametre);

int index\_of\_per = it - max\_in\_row.begin();

cout << "Переферийная вершина:" << index\_of\_per - n << endl;

auto it\_for = find(max\_in\_row.begin(), max\_in\_row.end(), \*radius);

int index\_of\_cent = it\_for - max\_in\_row.begin();

cout << "Центральная вершина:" << index\_of\_cent - n << endl;

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

{

delete[] matrix[i];

}

delete[] matrix;

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

//{

// delete[] incidence[i];

//}

//delete[] incidence;

system("pause>nul");

return 0;

}

vector<int> BFSD\_for\_weighted\_matrix(int s, int\* m[], int n, int edges) {

vector<int> dist(n, INT\_MAX);

queue<int> q;

q.push(s);

dist[s] = 0;

//cout << "Порядок обхода:" << endl;

int vis;

while (!q.empty()) {

vis = q.front();

// Print the current node

q.pop();

// For every adjacent vertex to the current vertex

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

if (m[i][vis] > 0 && dist[i] > dist[vis] + m[i][vis]) {

// Push the adjacent node to the queue

q.push(i);

dist[i] = dist[vis] + m[i][vis];

}

}

}

return dist;

}