# **Лабораторная работа №5**

## **Определение характеристик графов**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Определите размер графа *G*, используя матрицу смежности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

**Ответ задание 1:**

пример кода на C++, который генерирует матрицу смежности для неориентированного графа, выводит ее на экран и определяет размер графа, изолированные, концевые и доминирующие вершины:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <windows.h>

using namespace std;

class Graph {

public:

int vertices;

vector<vector<int>> adjacencyMatrix;

Graph(int V) {

vertices = V;

adjacencyMatrix.resize(V, vector<int>(V, 0));

}

void addEdge(int v, int w) {

adjacencyMatrix[v][w] = 1;

adjacencyMatrix[w][v] = 1; // Для неориентированного графа

}

void displayMatrix() {

setlocale(LC\_ALL, "rus");

cout << "Матрица смежности:" << endl;

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

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

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

}

cout << endl;

}

}

void findVertexTypes() {

vector<bool> isolated(vertices, true);

vector<bool> endVertices(vertices, false);

vector<bool> dominating(vertices, true);

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

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

if (adjacencyMatrix[i][j] == 1) {

isolated[i] = false;

isolated[j] = false;

endVertices[i] = true;

endVertices[j] = true;

dominating[j] = false;

}

}

}

setlocale(LC\_ALL, "rus");

cout << "\nИзолированные вершины: ";

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

if (isolated[i])

cout << i << " ";

}

setlocale(LC\_ALL, "rus");

cout << "\nКонечные вершины: ";

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

if (endVertices[i])

cout << i << " ";

}

setlocale(LC\_ALL, "rus");

cout << "\nДоминирующие вершины: ";

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

if (dominating[i])

cout << i << " ";

}

cout << endl;

}

};

int main() {

srand(time(0));

int size = rand() % 5 + 3; // Генерация случайного размера графа от 3 до 7

Graph graph(size);

// Генерация случайных ребер

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

int v = rand() % size;

int w = rand() % size;

graph.addEdge(v, w);

}

graph.displayMatrix();

setlocale(LC\_ALL, "rus");

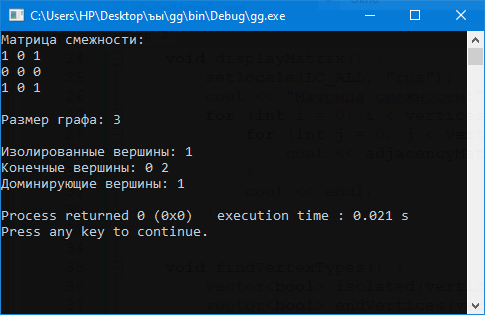
cout << "\nРазмер графа: " << size << endl;

graph.findVertexTypes();

return 0;

}

Этот код создает неориентированный граф с случайными ребрами, выводит матрицу смежности и определяет изолированные, концевые и доминирующие вершины.

****

**Задание 2\***

1. Постройте для графа G матрицу инцидентности.
2. Определите размер графа *G*, используя матрицу инцидентности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

**Ответ задание 2:**

**#include <iostream>**

**#include <vector>**

**#include <windows.h>**

**using namespace std;**

**class Graph {**

**public:**

**int vertices;**

**vector<vector<int>> adjacencyMatrix;**

**Graph(int V) : vertices(V), adjacencyMatrix(V, vector<int>(V, 0)) {}**

**void addEdge(int v, int w) {**

**adjacencyMatrix[v][w] = 1;**

**adjacencyMatrix[w][v] = 1; // Для неориентированного графа**

**}**

**void printIncidenceMatrix() {**

**int edges = 0;**

**setlocale(LC\_ALL, "rus");**

**for (int i = 0; i < vertices; ++i) {**

**for (int j = i + 1; j < vertices; ++j) {**

**if (adjacencyMatrix[i][j] == 1) {**

**edges++;**

**}**

**}**

**}**

**vector<vector<int>> incidenceMatrix(vertices, vector<int>(edges, 0));**

**int edgeIndex = 0;**

**for (int j = 0; j < vertices; ++j) {**

**for (int i = 0; i < vertices; ++i) {**

**if (adjacencyMatrix[i][j] == 1) {**

**incidenceMatrix[i][edgeIndex] = -1;**

**incidenceMatrix[j][edgeIndex] = 1;**

**edgeIndex++;**

**}**

**}**

**}**

**// Вывод матрицы инцидентности**

**cout << "матрицы инцидентности:" << endl;**

**for (const auto &row : incidenceMatrix) {**

**for (int val : row) {**

**cout << val << " ";**

**}**

**cout << endl;**

**}**

**// Определение размера графа**

**cout << "размер графа: " << vertices << endl;**

**// Поиск концевых вершин**

**cout << "Terminal vertices: ";**

**for (int i = 0; i < vertices; ++i) {**

**int sum = 0;**

**for (int j = 0; j < vertices; ++j) {**

**sum += adjacencyMatrix[i][j];**

**}**

**if (sum == 1) {**

**cout << i << " ";**

**}**

**}**

**cout << endl;**

**// Поиск изолированных вершин**

**cout << "изолированных вершин: ";**

**for (int i = 0; i < vertices; ++i) {**

**int sum = 0;**

**for (int j = 0; j < vertices; ++j) {**

**sum += adjacencyMatrix[i][j];**

**}**

**if (sum == 0) {**

**cout << i << " ";**

**}**

**}**

**cout << endl;**

**// Поиск доминирующих вершин**

**cout << "доминирующих вершин: ";**

**for (int i = 0; i < vertices; ++i) {**

**bool dominates = true;**

**for (int j = 0; j < vertices; ++j) {**

**if (i != j && adjacencyMatrix[i][j] == 0) {**

**dominates = false;**

**break;**

**}**

**}**

**if (dominates) {**

**cout << i << " ";**

**}**

**}**

**cout << endl;**

**}**

**};**

**int main() {**

**Graph g(4); // Пример с графом из 4 вершин**

**g.addEdge(0, 1);**

**g.addEdge(0, 2);**

**g.addEdge(1, 3);**

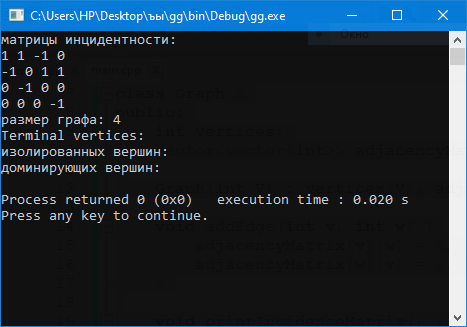
**g.addEdge(2, 3);**

**// Вывод матрицы инцидентности и характеристик графа**

**g.printIncidenceMatrix();**

**return 0;**

**}**

****