Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему: «Определение характеристик графов»

Выполнил:

студент группы 23ВВВ4

Соснин Глеб

Проверил:

доцент, Юрова О. В.

Пенза, 2024

**Цель** – научиться создавать графы, определять их характеристики, реализовывать на языке программирования C++ алгоритмы работы с графами.

**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.

**Листинг программы:**

#include <iostream>

#include <locale>

#include <vector>

#include <ctime>

#include <cstdlib>

using namespace std;

// Функция для генерации случайной матрицы смежности

vector<vector<int>> generate\_adjacency\_matrix(int size) {

vector<vector<int>> matrix(size, vector<int>(size, 0));

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

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

int edge = rand() % 2;

matrix[i][j] = edge;

matrix[j][i] = edge;

}

}

return matrix;

}

// Функция для анализа вершин

void analyze\_vertices(const vector<vector<int>>& matrix) {

int size = matrix.size();

vector<int> isolated, terminal, dominant;

cout << "Степени вершин (deg(v\_i)):" << endl;

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

int degree = 0;

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

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

degree++;

}

}

// Вывод степени вершины

cout << "deg(v\_" << i << ") = " << degree << endl;

if (degree == 0) {

isolated.push\_back(i);

}

else if (degree == 1) {

terminal.push\_back(i);

}

else if (degree == size - 1) {

dominant.push\_back(i);

}

}

// Вывод результатов

cout << "Изолированные вершины: ";

for (int v : isolated) cout << v << " ";

cout << endl;

cout << "Концевые вершины: ";

for (int v : terminal) cout << v << " ";

cout << endl;

cout << "Доминирующие вершины: ";

for (int v : dominant) cout << v << " ";

cout << endl;

}

// Функция для вывода матрицы смежности

void print\_matrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int elem : row) {

cout << elem << " ";

}

cout << endl;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(static\_cast<unsigned int>(time(0)));

int size;

cout << "Введите количество вершин графа: ";

cin >> size;

vector<vector<int>> adjacency\_matrix = generate\_adjacency\_matrix(size);

// Вывод матрицы смежности

cout << "Матрица смежности графа:" << endl;

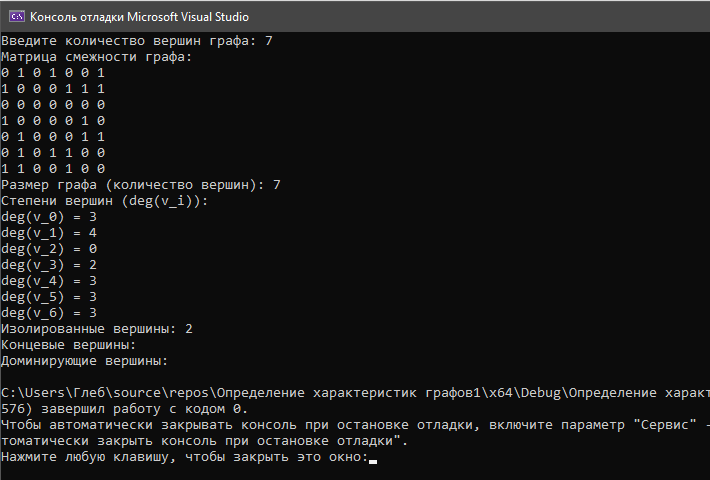
print\_matrix(adjacency\_matrix);

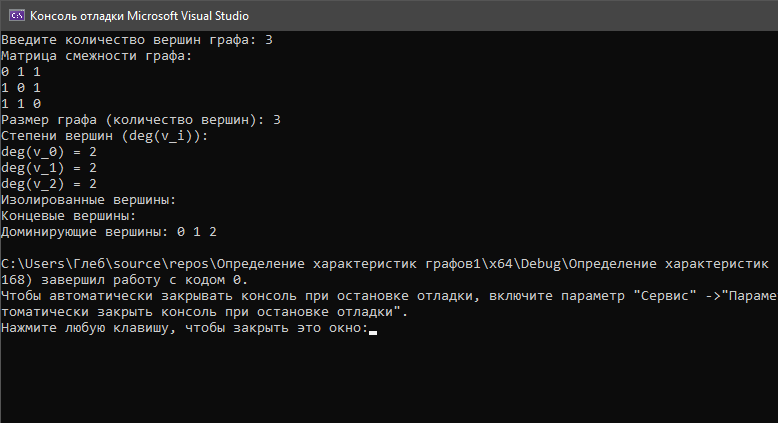
cout << "Размер графа (количество вершин): " << size << endl;

analyze\_vertices(adjacency\_matrix);

return 0;

}





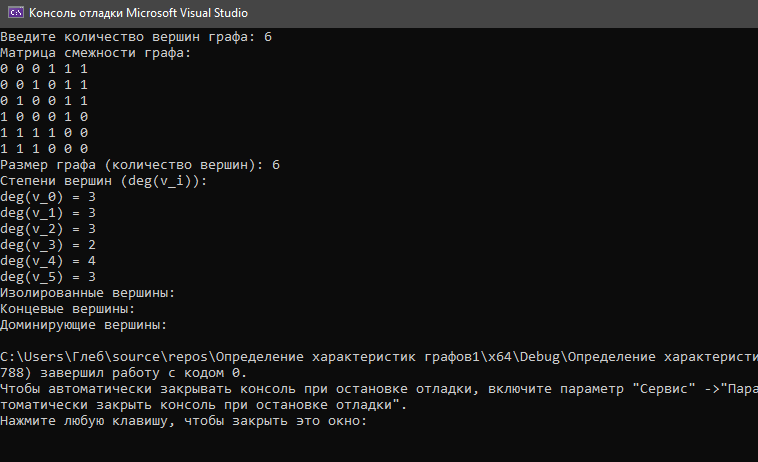


Рисунок 1, 2, 3 – Матрица смежности графов, анализ вершин на этих графах

Степенью вершины графа G называется число инцидентных ей ребер. Степень вершины *vi* обозначается через *deg(vi).*

Функции *analyze\_vertices()* выводит степень каждой вершины.

Функция *generate\_adjacency*\_*matrix(int size):* создает матрицу смежности для графа заданного размера. Использует случайные значения 0 и 1 для заполнения верхнего треугольника матрицы и симметрично копирует их в нижний треугольник, так как граф неориентированный.

Функция *analyze\_vertices(const vector<vector<int>>& matrix):* перебирает вершины, вычисляет их степень и относит их к одной из категорий:

* Изолированные (нет рёбер).
* Концевые (имеют ровно одно ребро).
* Доминирующие (связаны со всеми остальными вершинами).

Функция *print\_matrix(const vector<vector<int>>& matrix):* просто выводит матрицу смежности в консоль.

Функция *main():* запрашивает количество вершин графа, вызывает функцию генерации матрицы, выводит её, затем выполняет анализ вершин и выводит результаты.

**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.

**Листинг программы:**

#include <iostream>

#include <locale>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

// Функция для генерации случайного неориентированного графа

vector<pair<int, int>> generate\_edges(int size) {

vector<pair<int, int>> edges;

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

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

if (rand() % 2) {

edges.push\_back({ i, j });

}

}

}

return edges;

}

vector<vector<int>> generate\_incidence\_matrix(int vertices, const vector<pair<int, int>>& edges) {

vector<vector<int>> incidence\_matrix(vertices, vector<int>(edges.size(), 0));

for (int e = 0; e < edges.size(); ++e) {

int u = edges[e].first;

int v = edges[e].second;

incidence\_matrix[u][e] = 1;

incidence\_matrix[v][e] = 1;

}

return incidence\_matrix;

}

void analyze\_vertices(const vector<vector<int>>& incidence\_matrix) {

int vertices = incidence\_matrix.size();

int edges = incidence\_matrix[0].size();

vector<int> isolated, terminal, dominant;

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

int degree = 0; // Степень вершины

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

if (incidence\_matrix[i][j] == 1) {

degree++;

}

}

if (degree == 0) {

isolated.push\_back(i);

}

else if (degree == 1) {

terminal.push\_back(i);

}

else if (degree == edges) {

dominant.push\_back(i);

}

}

// Вывод результатов

cout << "Изолированные вершины: ";

for (int v : isolated) cout << v << " ";

cout << endl;

cout << "Концевые вершины: ";

for (int v : terminal) cout << v << " ";

cout << endl;

cout << "Доминирующие вершины: ";

for (int v : dominant) cout << v << " ";

cout << endl;

}

void print\_matrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int elem : row) {

cout << elem << " ";

}

cout << endl;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(static\_cast<unsigned int>(time(0)));

int vertices;

cout << "Введите количество вершин графа: ";

cin >> vertices;

vector<pair<int, int>> edges = generate\_edges(vertices);

vector<vector<int>> incidence\_matrix = generate\_incidence\_matrix(vertices, edges);

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

print\_matrix(incidence\_matrix);

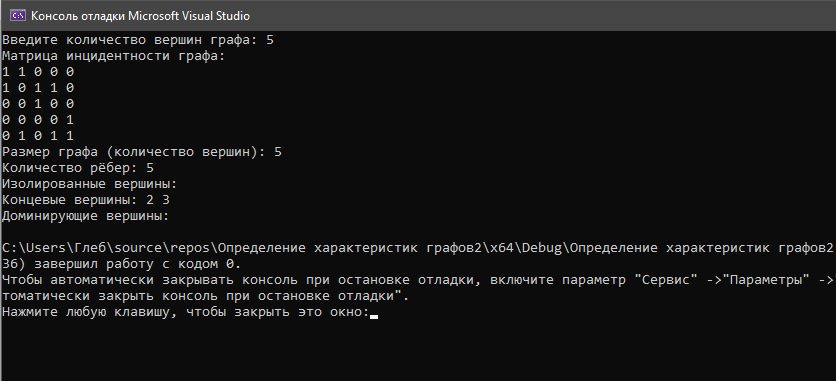
cout << "Размер графа (количество вершин): " << vertices << endl;

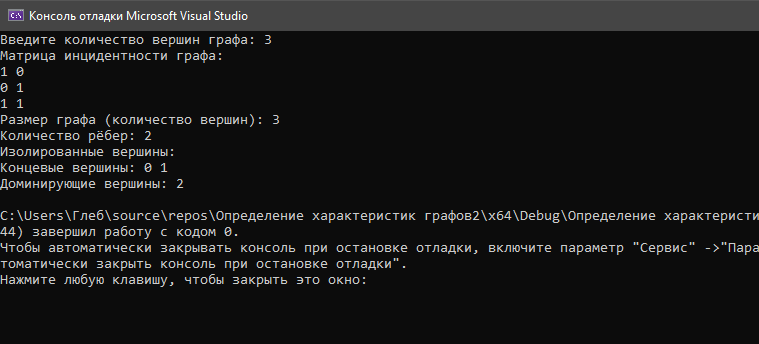
cout << "Количество рёбер: " << edges.size() << endl;

analyze\_vertices(incidence\_matrix);

return 0;

}





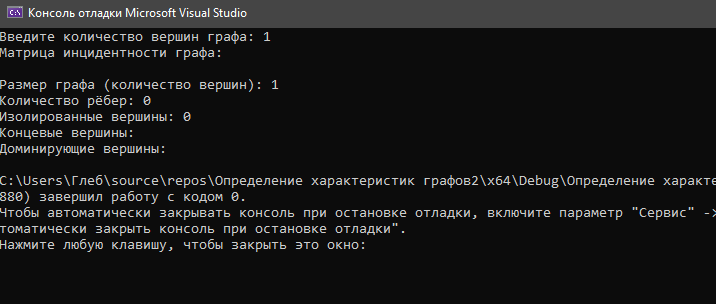


Рисунок 4, 5, 6 – Матрица инцидентности графов, анализ вершин, подсчет количества ребер

Функция *generate\_edges():* генерирует список рёбер для неориентированного графа. Ребра соединяют случайные вершины.

Функция *generate\_incidence\_matrix():* построение матрицы инцидентности на основе списка рёбер. Для каждого ребра 𝑒 обновляются строки двух вершин, инцидентных этому ребру.

Функция *analyze\_vertices():* определяет изолированные, концевые и доминирующие вершины на основе матрицы инцидентности:

* Изолированная вершина не инцидентна ни одному ребру.
* Концевая вершина инцидентна ровно одному ребру.
* Доминирующая вершина инцидентна всем рёбрам.

Функция *main():* запрашивает количество вершин, генерирует рёбра и матрицу инцидентности, выводит её и анализирует вершины.

**Вывод:** научились создавать графы, определять их характеристики, реализовывать на языке программирования C++ алгоритмы работы с графами.