Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 5

по дисциплине: "Логика и ОА в ИЗ"

## на тему: "Определение характеристик графов"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза, 2023

**Цель:**

**Научиться генерировать матрицу смежности, и с помощью неё создавать матрицу инцидентности, а так же считать размерность графа и искать изолированные, концевые и доминирующие вершины.**

**Общие сведения.**

Если *G –* граф (рисунок 1), содержащий непустое множество *n* вершин *V* и множество ребер *E,* где *e(vi, vj) –* ребро между двумя произвольными вершинами *vi* и*vj*, тогда **размер** графа G есть мощность множества ребер |E(G)| или, количество ребер графа.

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



Рисунок 1 – Граф

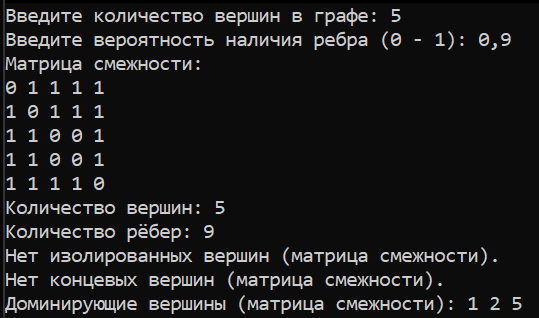
Вершина *vi* со степенью 0 называется **изолированной***,* со степенью 1 – **концевой***.*

Вершина графа, смежная с каждой другой его вершиной, называется **доминирующей***.*

**Практическая часть.**

### **Задание 1**

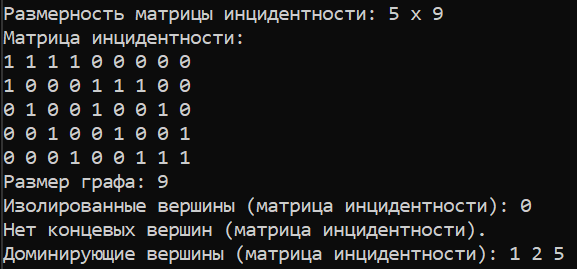
1. Сгенерировали (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Вывели матрицу на экран.



1. Определили размер графа *G*, используя матрицу смежности графа (В данном примере = 9).
2. Нашли изолированные (0), концевые (0) и доминирующие вершины (1,2,5).

### **Задание 2\***

1. Постройте для графа G матрицу инцидентности.



1. Определили размер графа *G*, используя матрицу инцидентности графа. (9)
2. Нашли изолированные (0), концевые (0) и доминирующие вершины (1,2,5).

**Вывод:**

**Мы научились генерировать матрицу смежности, и с помощью неё создавать матрицу инцидентности, а также считать размерность графа и искать изолированные, концевые и доминирующие вершины.**

**Приложение А (Листинг)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <locale.h>

using namespace std;

// Функция для поиска изолированных вершин (рабоnf с матрицей смежности)

vector<int> findIsolatedVertices(const vector<vector<int>>& adjacency\_matrix) {

vector<int> isolated\_vertices;

for (int i = 1; i < adjacency\_matrix.size(); ++i) {

bool isolated = true;

for (int j = 1; j < adjacency\_matrix[i].size(); ++j) {

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

isolated = false;

break;

}

}

if (isolated) {

isolated\_vertices.push\_back(i);

}

}

return isolated\_vertices;

}

// Функция для поиска концевых вершин (работа с матрицей смежности)

vector<int> findLeafVertices(const vector<vector<int>>& adjacency\_matrix) {

vector<int> leaf\_vertices;

for (int i = 1; i < adjacency\_matrix.size(); ++i) {

int degree = 0;

for (int j = 1; j < adjacency\_matrix[i].size(); ++j) {

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

degree++;

}

}

if (degree == 1) {

leaf\_vertices.push\_back(i);

}

}

return leaf\_vertices;

}

// Функция для поиска доминирующих вершин (работа с матрицей смежности)

vector<int> findDominatingVertices(const vector<vector<int>>& adjacency\_matrix) {

vector<int> dominating\_vertices;

for (int i = 1; i < adjacency\_matrix.size(); ++i) {

bool dominating = true;

for (int j = 1; j < adjacency\_matrix[i].size(); ++j) {

if (i != j && adjacency\_matrix[i][j] != 1) {

dominating = false;

break;

}

}

if (dominating) {

dominating\_vertices.push\_back(i);

}

}

return dominating\_vertices;

}

// Функция для создания и вывода матрицы инцидентности

vector < vector < int>> createAndPrintIncidenceMatrix(const vector < vector < int>>& adjacency\_matrix, int num\_edges) {

vector < vector < int>> incidence\_matrix(adjacency\_matrix.size(), vector<int>(num\_edges, 0));

int edge\_idx = 0;

for (int i = 1; i < adjacency\_matrix.size(); ++i) {

for (int j = i + 1; j < adjacency\_matrix[i].size(); ++j) {

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

incidence\_matrix[i][edge\_idx] = 1;

incidence\_matrix[j][edge\_idx] = 1;

edge\_idx++;

}

}

}

printf("Матрица инцидентности:\n");

for (int i = 1; i < incidence\_matrix.size(); ++i) {

for (int j = 0; j < num\_edges; ++j) {

printf("%d ", incidence\_matrix[i][j]);

}

printf("\n");

}

return incidence\_matrix;

}

int getGraphSizeFromIncidenceMatrix(const vector < vector < int>>& incidence\_matrix) {

// Возвращает количество столбцов в матрице инцидентности, что является размером графа

return incidence\_matrix[0].size(); // предполагается, что матрица не пуста

}

void printIncidenceMatrixDimensions(int num\_vertices, int num\_edges) {

printf("Размерность матрицы инцидентности: %d x %d\n", num\_vertices, num\_edges);

}

//Функция для поиска изолированных вершин на основе матрицы инцидентности

vector<int> findIsolatedVerticesIncidence(const vector < vector < int>>& incidence\_matrix) {

vector<int> isolated\_vertices;

for (int i = 0; i < incidence\_matrix.size(); ++i) {

bool isolated = true;

for (int j = 0; j < incidence\_matrix[i].size(); ++j) {

if (incidence\_matrix[i][j] != 0) {

isolated = false;

break;

}

}

if (isolated) {

isolated\_vertices.push\_back(i);

}

}

return isolated\_vertices;

}

//Функция для поиска концевых вершин на основе матрицы инцидентности

vector<int> findLeafVerticesIncidence(const vector < vector < int>>& incidence\_matrix) {

vector<int> leaf\_vertices;

for (int i = 1; i < incidence\_matrix.size(); ++i) {

int degree = 0;

for (int j = 0; j < incidence\_matrix[i].size(); ++j) {

if (incidence\_matrix[i][j] != 0) {

degree++;

}

}

if (degree == 1) {

leaf\_vertices.push\_back(i);

}

}

return leaf\_vertices;

}

// Функция для поиска доминирующих вершин на основе матрицы инцидентности

vector<int> findDominatingVerticesIncidence(const vector < vector < int>>& incidence\_matrix) {

vector<int> dominating\_vertices;

int total\_vertices = incidence\_matrix.size();

for (int i = 1; i < incidence\_matrix.size(); ++i) {

int edge\_count = 0;

for (int j = 1; j < incidence\_matrix[i].size(); ++j) {

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

edge\_count++;

}

}

if (edge\_count == total\_vertices - 1) {

dominating\_vertices.push\_back(i);

}

}

return dominating\_vertices;

}

int main() {

setlocale(LC\_ALL, "RUS");

int num\_vertices;

double edge\_probability;

printf("Введите количество вершин в графе: ");

scanf("%d", &num\_vertices);

printf("Введите вероятность наличия ребра (0 - 1): ");

scanf("%lf", &edge\_probability);

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

vector<vector<int>> adjacency\_matrix(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

int num\_edges = 0;

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < edge\_probability) {

adjacency\_matrix[i][j] = 1;

adjacency\_matrix[j][i] = 1;

num\_edges++;

}

}

}

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", adjacency\_matrix[i][j]);

}

printf("\n");

}

printf("Количество вершин: %d\n", num\_vertices);

printf("Количество рёбер: %d\n", num\_edges);

vector<int> isolated\_vertices = findIsolatedVertices(adjacency\_matrix);

if (!isolated\_vertices.empty()) {

printf("Изолированные вершины (матрица смежности):");

for (int i = 0; i < isolated\_vertices.size(); ++i) {

printf(" %d", isolated\_vertices[i]);

}

printf("\n");

}

else {

printf("Нет изолированных вершин (матрица смежности).\n");

}

vector<int> leaf\_vertices = findLeafVertices(adjacency\_matrix);

if (!leaf\_vertices.empty()) {

printf("Концевые вершины (матрица смежности):");

for (int i = 0; i < leaf\_vertices.size(); ++i) {

printf(" %d", leaf\_vertices[i]);

}

printf("\n");

}

else {

printf("Нет концевых вершин (матрица смежности).\n");

}

vector<int> dominating\_vertices = findDominatingVertices(adjacency\_matrix);

if (!dominating\_vertices.empty()) {

printf("Доминирующие вершины (матрица смежности):");

for (int i = 0; i < dominating\_vertices.size(); ++i) {

printf(" %d", dominating\_vertices[i]);

}

printf("\n");

}

else {

printf("Нет доминирующих вершин (матрица смежности).\n\n\n");

}

// createAndPrintIncidenceMatrix(adjacency\_matrix, num\_edges);

printIncidenceMatrixDimensions(num\_vertices, num\_edges);

vector < vector < int>> incidence\_matrix = createAndPrintIncidenceMatrix(adjacency\_matrix, num\_edges);

vector<int> isolated\_vertices\_incidence = findIsolatedVerticesIncidence(incidence\_matrix);

// Определение размера графа

int graph\_size = getGraphSizeFromIncidenceMatrix(incidence\_matrix);

printf("Размер графа: %d\n", graph\_size);

if (!isolated\_vertices\_incidence.empty()) {

printf("Изолированные вершины (матрица инцидентности):");

for (int vertex : isolated\_vertices\_incidence) {

printf(" %d", vertex);

}

printf("\n");

}

else {

printf("Нет изолированных вершин (матрица инцидентности).\n");

}

vector<int> leaf\_vertices\_incidence = findLeafVerticesIncidence(incidence\_matrix);

if (!leaf\_vertices\_incidence.empty()) {

printf("Концевые вершины (матрица инцидентности):");

for (int vertex : leaf\_vertices\_incidence) {

printf(" %d", vertex);

}

printf("\n");

}

else {

printf("Нет концевых вершин (матрица инцидентности).\n");

}

if (!dominating\_vertices.empty()) {

printf("Доминирующие вершины (матрица инцидентности):");

for (int i = 0; i < dominating\_vertices.size(); ++i) {

printf(" %d", dominating\_vertices[i]);

}

printf("\n");

}

else {

printf("Нет доминирующих вершин (матрица инцидентности).\n\n\n");

}

return 0;

}