Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Основы алгоритмизации в инженерных задачах»

на тему «**Определение характеристик графов**»

Выполнил:

студент группы 22ВВП1

Воробьева М.М.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Цель:** Научиться представлять графы в виде матриц смежности и инцидентности. Определять основные характеристики графов.

**Лабораторное задание:**

**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.

**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.

Примечание: задания, помеченные символом \* выполняются по указанию преподавателя.

**Листинг программы.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <locale.h>

using namespace std;

// Функция для поиска изолированных вершин (матрицей смежности)

vector<int> findIsolatedVertices(const vector<vector<int>>& cmej\_mat) {

vector<int> isolated\_vertices;

for (int i = 1; i < cmej\_mat.size(); ++i) {

bool isolated = true;

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

if (cmej\_mat[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>>& cmej\_mat) {

vector<int> leaf\_vertices;

for (int i = 1; i < cmej\_mat.size(); ++i) {

int degree = 0;

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

if (cmej\_mat[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>>& cmej\_mat) {

vector<int> dominating\_vertices;

for (int i = 1; i < cmej\_mat.size(); ++i) {

bool dominating = true;

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

if (i != j && cmej\_mat[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>>& cmej\_mat, int colich\_reb) {

vector < vector < int>> incidence\_matrix(cmej\_mat.size(), vector<int>(colich\_reb, 0));

int edge\_idx = 0;

for (int i = 1; i < cmej\_mat.size(); ++i) {

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

if (cmej\_mat[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 < colich\_reb; ++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 col\_ver, int colich\_reb) {

printf("Размерность матрицы инцидентности: %d x %d\n", col\_ver, colich\_reb);

}

//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 col\_ver;

printf("Введите количество вершин: ");

scanf("%d", &col\_ver);

srand(time(NULL));

vector<vector<int>> cmej\_mat(col\_ver + 1, vector<int>(col\_ver + 1, 0));

int colich\_reb = 0;

for (int i = 1; i <= col\_ver; ++i) {

for (int j = i + 1; j <= col\_ver; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < 0.5) {

cmej\_mat[i][j] = 1;

cmej\_mat[j][i] = 1;

colich\_reb++;

}

}

}

printf("Матрица смежности:\n");

for (int i = 1; i <= col\_ver; ++i) {

for (int j = 1; j <= col\_ver; ++j) {

printf("%d ", cmej\_mat[i][j]);

}

printf("\n");

}

printf("Количество вершин: %d\n", col\_ver);

printf("Количество рёбер: %d\n", colich\_reb);

vector<int> isolated\_vertices = findIsolatedVertices(cmej\_mat);

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(cmej\_mat);

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(cmej\_mat);

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");

}

printIncidenceMatrixDimensions(col\_ver, colich\_reb);

vector < vector < int>> incidence\_matrix = createAndPrintIncidenceMatrix(cmej\_mat, colich\_reb);

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;

}

**Результат работы программы:**



**Вывод:** в ходе выполнения лабораторной работы были освоены навыки представления графов в виде матриц смежности и инцидентности, определения основных характеристик графа.