Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Определение характеристик графов**.**»

Выполнили:

студенты группы 23ВВВ4

Брагин А.М.

Зарубин Я.

Герасимов К.

Приняли:

Деев М.В.

Юрова О.В.

Пенза 2024

**Общие сведения.**

Если G – граф (рисунок 1), содержащий непустое множество n вершин

V и множество ребер E, где e(v i , v j ) – ребро между двумя произвольными вершинами v i  и v j , 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 v i обозначается через deg(v i ).

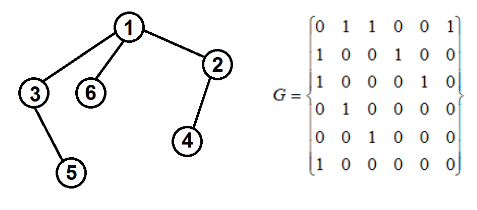


Рисунок 1 – Граф

Вершина v i со степенью 0 называется изолированной, со степенью 1 –

концевой.

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

**Практическая часть:**

**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.

Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void generateAdjacencyMatrix(int n, int \*\*graph) {

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

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

if (i == j) {

graph[i][j] = 0;

} else {

graph[i][j] = rand() % 2;

graph[j][i] = graph[i][j];

}

}

}

}

void printAdjacencyMatrix(int n, int \*\*graph) {

printf("Матрица смежности:\n");

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

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

printf("%d ", graph[i][j]);

}

printf("\n");

}

}

int calculateEdgesByAdjacencyMatrix(int n, int \*\*graph) {

int edgeCount = 0;

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

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

edgeCount += graph[i][j];

}

}

return edgeCount;

}

void findSpecialVerticesByAdjacencyMatrix(int n, int \*\*graph) {

int degree;

int isolated = 0, pendant = 0, dominating = 0;

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

degree = 0;

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

degree += graph[i][j];

}

if (degree == 0) {

printf("Вершина %d: изолированная (по матрице смежности)\n", i + 1);

isolated++;

} else if (degree == 1) {

printf("Вершина %d: концевая (по матрице смежности)\n", i + 1);

pendant++;

} else if (degree == n - 1) {

printf("Вершина %d: доминирующая (по матрице смежности)\n", i + 1);

dominating++;

}

}

printf("\nПо матрице смежности: изолированных вершин = %d, конечных = %d, доминирующих = %d\n", isolated, pendant, dominating);

}

void generateIncidenceMatrix(int n, int \*\*graph, int \*\*incidence, int \*edgeCount) {

int edgeIndex = 0;

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

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

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

incidence[i][edgeIndex] = 1;

incidence[j][edgeIndex] = 1;

edgeIndex++;

}

}

}

\*edgeCount = edgeIndex;

}

void printIncidenceMatrix(int n, int edgeCount, int \*\*incidence) {

printf("Матрица инцидентности:\n");

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

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

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

}

printf("\n");

}

}

int calculateEdgesByIncidenceMatrix(int edgeCount) {

return edgeCount;

}

void findSpecialVerticesByIncidenceMatrix(int n, int edgeCount, int \*\*incidence) {

int isolated = 0, pendant = 0, dominating = 0;

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

int degree = 0;

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

degree += incidence[i][j];

}

if (degree == 0) {

printf("Вершина %d: изолированная (по матрице инцидентности)\n", i + 1);

isolated++;

} else if (degree == 1) {

printf("Вершина %d: концевая (по матрице инцидентности)\n", i + 1);

pendant++;

} else if (degree == n - 1) {

printf("Вершина %d: доминирующая (по матрице инцидентности)\n", i + 1);

dominating++;

}

}

printf("\nПо матрице инцидентности: изолированных вершин = %d, конечных = %d, доминирующих = %d\n", isolated, pendant, dominating);

}

int main() {

srand(time(0));

int n;

printf("Введите количество вершин графа: ");

scanf("%d", &n);

if (n > 10) {

printf("Количество вершин не может превышать 10\n");

return 1;

}

int \*\*graph = (int \*\*)malloc(n \* sizeof(int \*));

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

graph[i] = (int \*)malloc(n \* sizeof(int));

}

int \*\*incidence = (int \*\*)malloc(n \* sizeof(int \*));

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

incidence[i] = (int \*)calloc(n \* (n - 1) / 2, sizeof(int));

}

int edgeCount;

generateAdjacencyMatrix(n, graph);

printAdjacencyMatrix(n, graph);

int edgesAdjacency = calculateEdgesByAdjacencyMatrix(n, graph);

printf("\nРазмер графа по матрице смежности: %d\n", edgesAdjacency);

generateIncidenceMatrix(n, graph, incidence, &edgeCount);

printIncidenceMatrix(n, edgeCount, incidence);

int edgesIncidence = calculateEdgesByIncidenceMatrix(edgeCount);

printf("\nРазмер графа по матрице инцидентности: %d\n", edgesIncidence);

// Поиск специальных вершин по матрице смежности

findSpecialVerticesByAdjacencyMatrix(n, graph);

// Поиск специальных вершин по матрице инцидентности

findSpecialVerticesByIncidenceMatrix(n, edgeCount, incidence);

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

free(graph[i]);

free(incidence[i]);

}

free(graph);

free(incidence);

return 0;

}

**Результат работы программы:**

