МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №5  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Определение характеристик графов»

Выполнил:

Студент группы 23ВВВ2

Чупраков С. В.

Пичаев И. А.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Приобрести навыки программирования и анализа графов с использованием матриц смежности и инцидентности. Генерировать случайные графы и представлять их в виде матриц. Анализировать свойства графов, такие как изолированные, концевые и доминирующие вершины. Понимать и применять различные представления графов, что является важным аспектом в теории графов и их практическом использовании.

**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.

**Ход работы**

**Задание 1.**

Создали функцию для генерации матрицы смежности и вывода ее на экран.

void generateAdjacencyMatrix(int n) {

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

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

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

}

// Инициализация матрицы нулями

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

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

adjacencyMatrix[i][j] = 0;

}

}

// Заполнение матрицы смежности

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

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

int edge = rand() % 2;

adjacencyMatrix[i][j] = edge;

adjacencyMatrix[j][i] = edge; // Для неориентированного графа

}

}

// Вывод матрицы на экран

printf("Матрица смежности для неориентированного графа:\n");

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

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

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

}

printf("\n");

}

// Подсчет рёбер

int edgeCount = countEdges(adjacencyMatrix, n);

printf("Количество рёбер в графе: %d\n", edgeCount);

// Нахождение вершин

findVertices(adjacencyMatrix, n);

// Освобождение памяти

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

free(adjacencyMatrix[i]);

}

free(adjacencyMatrix);

}

Для определения размера графа используется функция countEdges(int\*\* adjacencyMatrix, int n), которая вызывается в функции generateAdjacencyMatrix(int n).

int countEdges(int\*\* adjacencyMatrix, int n) {

int count = 0;

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

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

count += adjacencyMatrix[i][j];

}

}

return count;

}

Для поиска изолированных, концевых и доминирующих вершин используется функция findVertices(adjacencyMatrix, n); , которая также вызывается в функции generateAdjacencyMatrix(int n) .

void findVertices(int\*\* adjacencyMatrix, int n) {

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

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

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

int isolatedCount = 0, terminalCount = 0, dominatingCount = 0;

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

int degree = 0;

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

degree += adjacencyMatrix[i][j];

}

if (degree == 0) {

isolated[isolatedCount++] = i; // Изолированная вершина

}

else if (degree == 1) {

terminal[terminalCount++] = i; // Концевая вершина

}

// Проверка на доминирующую вершину

int isDominating = 1; // Предполагаем, что вершина доминирующая

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

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

isDominating = 0; // Если есть хотя бы одна не смежная вершина

break;

}

}

if (isDominating) {

dominating[dominatingCount++] = i; // Доминирующая вершина

}

}

// Вывод результатов

printf("Изолированные вершины: ");

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

printf("%d ", isolated[i]+1);

}

printf("\n");

printf("Концевые вершины: ");

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

printf("%d ", terminal[i]+1);

}

printf("\n");

printf("Доминирующие вершины: ");

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

printf("%d ", dominating[i]+1);

}

printf("\n");

// Освобождение памяти

free(isolated);

free(terminal);

free(dominating);

}

**Задание 2:**

Создали функцию для генерации матрицы инцидентности и вывода ее на экран на основе матрицы смежности.

void generateIncidenceMatrix(int n, int\*\* adjacencyMatrix) {

int edgeCount = countEdges(adjacencyMatrix, n);

// Создание матрицы инцидентности с количеством вершин в строках и количеством рёбер в столбцах

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

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

incidenceMatrix[i] = (int\*)malloc(edgeCount \* sizeof(int));

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

incidenceMatrix[i][j] = 0; // Инициализация нулями

}

}

int edgeIndex = 0;

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

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

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

incidenceMatrix[i][edgeIndex] = 1; // Инцидентность вершины i

incidenceMatrix[j][edgeIndex] = 1; // Инцидентность вершины j

edgeIndex++;

}

}

}

// Вывод матрицы инцидентности, теперь рёбра в столбцах

printf("Матрица инцидентности:\n");

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

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

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

}

printf("\n");

}

printf("\nРазмер графа определяется количеством столбов в матрице инцинтентности. \nРазмер графа: %d\n", edgeCount);

findVerticesByIncidenceMatrix(n, incidenceMatrix, edgeCount);

// Освобождение памяти

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

free(incidenceMatrix[i]);

}

free(incidenceMatrix);

}

Для поиска изолированных, концевых и доминирующих вершин используется функция findVerticesByIncidenceMatrix().

void findVerticesByIncidenceMatrix(int n, int\*\* incidenceMatrix, int edgeCount) {

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

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

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

int isolatedCount = 0, terminalCount = 0, dominatingCount = 0;

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

int degree = 0;

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

degree += incidenceMatrix[i][j];

}

if (degree == 0) {

isolated[isolatedCount++] = i; // Изолированная вершина

}

else if (degree == 1) {

terminal[terminalCount++] = i; // Концевая вершина

}

// Проверка на доминирующую вершину

int isDominating = 1; // Предполагаем, что вершина доминирующая

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

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

isDominating = 0; // Если есть хотя бы одна не смежная вершина

break;

}

}

if (isDominating) {

dominating[dominatingCount++] = i; // Доминирующая вершина

}

}

// Вывод результатов

printf("Изолированные вершины: ");

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

printf("%d ", isolated[i] + 1);

}

printf("\n");

printf("Концевые вершины: ");

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

printf("%d ", terminal[i] + 1);

}

printf("\n");

printf("Доминирующие вершины: ");

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

printf("%d ", dominating[i] + 1);

}

printf("\n");

// Освобождение памяти

free(isolated);

free(terminal);

free(dominating);

}

### Результаты работы программы

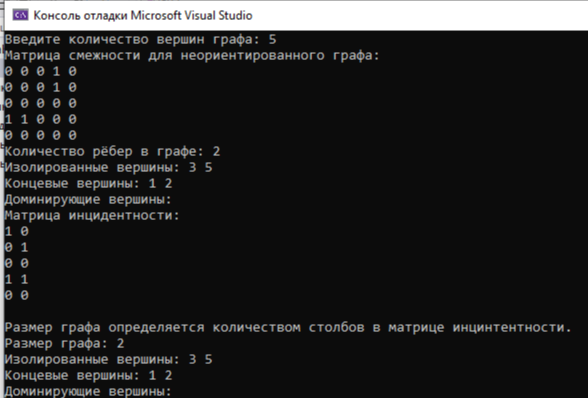


Рисунок 1 — Результаты работы программы

### Вывод

В результате выполнения работы были успешно разработаны и реализованы функции работы c графами: создание матриц смежности и матриц инцидентности, определение размера графа по матрице, поиск изолированных, концевых и доминирующих вершин.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

void generateAdjacencyMatrix(int n);

int countEdges(int\*\* adjacencyMatrix, int n);

void findVertices(int\*\* adjacencyMatrix, int n);

void generateIncidenceMatrix(int n, int\*\* adjacencyMatrix);

void findVerticesByIncidenceMatrix(int n, int\*\* incidenceMatrix, int edgeCount);

void generateAdjacencyMatrix(int n) {

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

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

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

}

// Инициализация матрицы нулями

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

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

adjacencyMatrix[i][j] = 0;

}

}

// Заполнение матрицы смежности

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

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

int edge = rand() % 2;

adjacencyMatrix[i][j] = edge;

adjacencyMatrix[j][i] = edge; // Для неориентированного графа

}

}

// Вывод матрицы на экран

printf("Матрица смежности для неориентированного графа:\n");

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

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

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

}

printf("\n");

}

// Подсчет рёбер

int edgeCount = countEdges(adjacencyMatrix, n);

printf("Количество рёбер в графе: %d\n", edgeCount);

// Нахождение вершин

findVertices(adjacencyMatrix, n);

generateIncidenceMatrix(n, adjacencyMatrix);

// Освобождение памяти

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

free(adjacencyMatrix[i]);

}

free(adjacencyMatrix);

}

int countEdges(int\*\* adjacencyMatrix, int n) {

int count = 0;

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

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

count += adjacencyMatrix[i][j];

}

}

return count;

}

void findVertices(int\*\* adjacencyMatrix, int n) {

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

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

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

int isolatedCount = 0, terminalCount = 0, dominatingCount = 0;

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

int degree = 0;

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

degree += adjacencyMatrix[i][j];

}

if (degree == 0) {

isolated[isolatedCount++] = i; // Изолированная вершина

}

else if (degree == 1) {

terminal[terminalCount++] = i; // Концевая вершина

}

// Проверка на доминирующую вершину

int isDominating = 1; // Предполагаем, что вершина доминирующая

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

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

isDominating = 0; // Если есть хотя бы одна не смежная вершина

break;

}

}

if (isDominating) {

dominating[dominatingCount++] = i; // Доминирующая вершина

}

}

// Вывод результатов

printf("Изолированные вершины: ");

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

printf("%d ", isolated[i]+1);

}

printf("\n");

printf("Концевые вершины: ");

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

printf("%d ", terminal[i]+1);

}

printf("\n");

printf("Доминирующие вершины: ");

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

printf("%d ", dominating[i]+1);

}

printf("\n");

// Освобождение памяти

free(isolated);

free(terminal);

free(dominating);

}

void findVerticesByIncidenceMatrix(int n, int\*\* incidenceMatrix, int edgeCount) {

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

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

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

int isolatedCount = 0, terminalCount = 0, dominatingCount = 0;

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

int degree = 0;

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

degree += incidenceMatrix[i][j];

}

if (degree == 0) {

isolated[isolatedCount++] = i; // Изолированная вершина

}

else if (degree == 1) {

terminal[terminalCount++] = i; // Концевая вершина

}

// Проверка на доминирующую вершину

int isDominating = 1; // Предполагаем, что вершина доминирующая

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

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

isDominating = 0; // Если есть хотя бы одна не смежная вершина

break;

}

}

if (isDominating) {

dominating[dominatingCount++] = i; // Доминирующая вершина

}

}

// Вывод результатов

printf("Изолированные вершины: ");

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

printf("%d ", isolated[i] + 1);

}

printf("\n");

printf("Концевые вершины: ");

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

printf("%d ", terminal[i] + 1);

}

printf("\n");

printf("Доминирующие вершины: ");

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

printf("%d ", dominating[i] + 1);

}

printf("\n");

// Освобождение памяти

free(isolated);

free(terminal);

free(dominating);

}

void generateIncidenceMatrix(int n, int\*\* adjacencyMatrix) {

int edgeCount = countEdges(adjacencyMatrix, n);

// Создание матрицы инцидентности с количеством вершин в строках и количеством рёбер в столбцах

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

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

incidenceMatrix[i] = (int\*)malloc(edgeCount \* sizeof(int));

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

incidenceMatrix[i][j] = 0; // Инициализация нулями

}

}

int edgeIndex = 0;

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

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

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

incidenceMatrix[i][edgeIndex] = 1; // Инцидентность вершины i

incidenceMatrix[j][edgeIndex] = 1; // Инцидентность вершины j

edgeIndex++;

}

}

}

// Вывод матрицы инцидентности, теперь рёбра в столбцах

printf("Матрица инцидентности:\n");

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

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

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

}

printf("\n");

}

printf("\nРазмер графа определяется количеством столбов в матрице инцинтентности. \nРазмер графа: %d\n", edgeCount);

findVerticesByIncidenceMatrix(n, incidenceMatrix, edgeCount);

// Освобождение памяти

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

free(incidenceMatrix[i]);

}

free(incidenceMatrix);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand((unsigned int)time(0)); // Инициализация генератора случайных чисел

int n;

printf("Введите количество вершин графа: ");

scanf("%d", &n);

generateAdjacencyMatrix(n);

return 0;

}