Министерство 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»

Выполнил:

Студент группы 23ВВВ2

Стрельцов А.П.

Федоров Б.М.

Приняли:

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Цель работы:** поработать с графами, выполнить задания и усвоить материал.

**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.

**ЛИСТИНГ**

**laba 5-1.cpp**

#include <stdio.h>

#include <random>

#include <Windows.h>

#include <conio.h>

#include <malloc.h>

#include <time.h>

int main(void) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//Задание 1.1

srand(time(NULL));

int n = 10, size = 0, size2 = 0, \* deg = NULL;

printf("Введите количество вершин: ");

scanf\_s("%d", &n);

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

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

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

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

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

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

G[j][i] = G[i][j];

if (i == j)

G[i][j] = 0;

}

}

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

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

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

}

printf("\n");

}

//Задание 1.2

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

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

size2 += G[i][j];

if (G[i][j] != 0)

size++;

}

}

size /= 2;

printf("Размер графа G: %d\n", size);

//Задание 1.3

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

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

deg[i] = 0;

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

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

if (G[i][j] != 0)

deg[i]++;

}

}

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

printf("Степень вершины %d = %d\n", i, deg[i]);

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

if (deg[i] == 0) printf("Вершина %d - изолированная\n", i);

if (deg[i] == 1) printf("Вершина %d - концевая\n", i);

if (deg[i] == n - 1) printf("Вершина %d - доминирующая\n", i);

}

return 0;

}

Результат работы программы

На рисунке 1 показана реализация задания №1.



Рисунок 1 - Результат работы программы №1.

**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**

**laba 5-2.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define MAX\_VERTICES 10 // Увеличим максимальное количество вершин для большей гибкости

#define MAX\_EDGES 20 // Увеличим максимальное количество рёбер

// Функция для создания матрицы инцидентности на основе матрицы смежности

void create\_incidence\_matrix(int graph[MAX\_VERTICES][MAX\_VERTICES], int incidence\_matrix[MAX\_VERTICES][MAX\_EDGES], int\* edges) {

int edge\_index = 0;

// Инициализация матрицы инцидентности

for (int i = 0; i < MAX\_VERTICES; i++) {

for (int j = 0; j < MAX\_EDGES; j++) {

incidence\_matrix[i][j] = 0;

}

}

// Заполнение матрицы инцидентности на основе матрицы смежности

for (int i = 0; i < MAX\_VERTICES; i++) {

for (int j = i + 1; j < MAX\_VERTICES; j++) {

if (graph[i][j] == 1) { // Если есть ребро

incidence\_matrix[i][edge\_index] = 1; // Инцидентность для первой вершины

incidence\_matrix[j][edge\_index] = 1; // Инцидентность для второй вершины

edge\_index++;

}

}

}

\*edges = edge\_index; // Обновляем количество рёбер

}

// Функция для нахождения изолированных, концевых и доминирующих вершин

void find\_special\_vertices(int incidence\_matrix[MAX\_VERTICES][MAX\_EDGES], int vertices, int edges) {

setlocale(LC\_ALL, "RUS");

int max\_incident\_count = 0;

int dominant\_vertex = -1;

// Сначала находим максимальное количество инцидентных рёбер

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

int incident\_count = 0;

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

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

incident\_count++;

}

}

if (incident\_count > max\_incident\_count) {

max\_incident\_count = incident\_count;

dominant\_vertex = i; // Запоминаем вершину с максимальным количеством рёбер

}

}

// Теперь определяем типы вершин

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

int incident\_count = 0;

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

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

incident\_count++;

}

}

if (incident\_count == 0) {

printf("Вершина %d является изолированной.\n", i);

}

else if (incident\_count == 1) {

printf("Вершина %d является концевой.\n", i);

}

else if (i == dominant\_vertex && incident\_count == max\_incident\_count) {

printf("Вершина %d является доминирующей.\n", i);

}

}

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL)); // Инициализация генератора случайных чисел

int num\_vertices;

printf("Введите количество вершин (максимум %d): ", MAX\_VERTICES);

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

// Проверка на допустимое количество вершин

if (num\_vertices < 1 || num\_vertices > MAX\_VERTICES) {

printf("Ошибка: количество вершин должно быть от 1 до %d.\n", MAX\_VERTICES);

return 1;

}

int graph[MAX\_VERTICES][MAX\_VERTICES] = { 0 }; // Инициализация графа нулями

int incidence\_matrix[MAX\_VERTICES][MAX\_EDGES];

int edges = 0;

// Генерация случайных рёбер

for (int i = 0; i < num\_vertices; i++) {

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

if (rand() % 2 && edges < MAX\_EDGES) { // С вероятностью 50% добавляем ребро, если не превышен лимит

graph[i][j] = 1;

graph[j][i] = 1; // Для неориентированного графа

edges++;

}

}

}

// Создание матрицы инцидентности на основе матрицы смежности

create\_incidence\_matrix(graph, incidence\_matrix, &edges);

// Вывод матрицы смежности

printf("Матрица смежности:\n");

for (int i = 0; i < num\_vertices; i++) {

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

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

}

printf("\n");

}

// Вывод матрицы инцидентности

printf("Матрица инцидентности:\n");

for (int i = 0; i < num\_vertices; i++) {

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

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

}

printf("\n");

}

// Нахождение специальных вершин

find\_special\_vertices(incidence\_matrix, num\_vertices, edges);

return 0;

}

Результат работы программы

На рисунке 2 показана реализация задания №2.

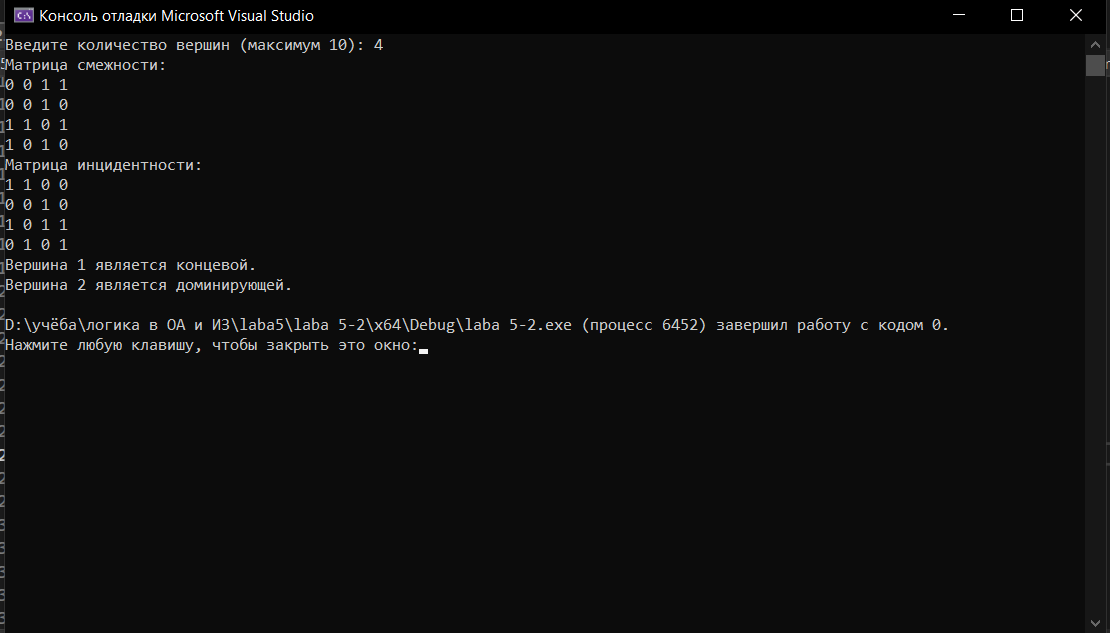


Рисунок 2 - Результат работы программы №2.

**Вывод**

Познакомились с графами, изучили новые функции и применили их в лабораторной работе.