Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

по лабораторной работе №5

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “Определение характеристик графов”

Выполнили:

Студенты группы 24ВВВ4

Васин И.В.

Анисимов А.Ю.

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2025

**Название:**

Определение характеристик графов

**Цель работы:**

Разработать программу, где реализуется поиск того или иного значения с клавиатуры

**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.

**Результат работы программы:**



**Выводы:**

В ходе выполнения лабораторной работы были разработаны программы, выполняющие указанные в лабораторной работе задачи. Результаты работ программ совпали с результатами трассировок, следовательно, программы работают без ошибок.

Получили опыт в создании проектов в среде MicrosoftVisualStudio.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int main() {

setlocale(LC\_ALL, "");

int N; // размерность матрицы

double p; // вероятность появления ребра

double p\_loop; // вероятность появления петли

printf("Введите размерность матрицы (N): ");

scanf("%d", &N);

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

scanf("%lf", &p);

printf("Введите вероятность появления петли (0,0 - 1,0): ");

scanf("%lf", &p\_loop);

// Проверка корректности ввода

if (N <= 0) {

printf("Ошибка: размерность должна быть положительным числом\n");

return 1;

}

if (p < 0.0 || p > 1.0) {

printf("Ошибка: вероятность должна быть в диапазоне от 0,0 до 1,0\n");

return 1;

}

if (p\_loop < 0.0 || p\_loop > 1.0) {

printf("Ошибка: вероятность петли должна быть в диапазоне от 0,0 до 1,0\n");

return 1;

}

// Выделение памяти для матрицы смежности

int\*\* adj\_matrix = (int\*\*)malloc(N \* sizeof(int\*));

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

adj\_matrix[i] = (int\*)malloc(N \* sizeof(int));

}

srand(time(NULL));

printf("\n=== ЗАДАНИЕ 1 ===\n\n");

// 1. Генерация матрицы смежности

printf("1. Матрица смежности:\n");

// Сначала заполняем всю матрицу нулями

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

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

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

}

}

// Генерируем петли (диагональные элементы)

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

double random\_value = (double)rand() / RAND\_MAX;

if (random\_value < p\_loop) {

adj\_matrix[i][i] = 1; // петля

}

}

// Генерируем ребра для недиагональных элементов

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

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

if (i != j) {

double random\_value = (double)rand() / RAND\_MAX;

if (random\_value < p) {

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

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

}

}

}

}

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

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

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

}

printf("\n");

}

int edges = 0;

int loops = 0;

printf("\n2. Степени вершин:\n");

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

int degree = 0;

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

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

if (i == j) {

degree += 2;

}

else {

degree += 1;

}

}

}

printf(" Вершина %d: степень %d", i, degree);

if (adj\_matrix[i][i] == 1) {

printf(" (есть петля)");

}

printf("\n");

}

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

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

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

edges++;

}

}

}

// Подсчет петель

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

if (adj\_matrix[i][i] == 1) {

loops++;

}

}

printf("\n Размер графа: %d ребер (из них %d петель)\n", edges + loops, loops);

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

int\* pendant = (int\*)malloc(N \* sizeof(int));

int\* dominant = (int\*)malloc(N \* sizeof(int));

int isolated\_count = 0, pendant\_count = 0, dominant\_count = 0;

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

int degree = 0;

int degree\_without\_loops = 0;

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

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

if (i == j) {

degree += 2;

}

else {

degree += 1;

degree\_without\_loops += 1;

}

}

}

if (degree\_without\_loops == 0) {

isolated[isolated\_count] = i;

isolated\_count++;

}

else if (degree\_without\_loops == 1) {

pendant[pendant\_count] = i;

pendant\_count++;

}

if (degree >= N) {

dominant[dominant\_count] = i;

dominant\_count++;

}

}

printf("\n3. Особые вершины:\n");

printf(" Изолированные: [");

for (int i = 0; i < isolated\_count; i++) {

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

if (i < isolated\_count - 1) printf(", ");

}

printf("]\n");

printf(" Концевые: [");

for (int i = 0; i < pendant\_count; i++) {

printf("%d", pendant[i]);

if (i < pendant\_count - 1) printf(", ");

}

printf("]\n");

printf(" Доминирующие: [");

for (int i = 0; i < dominant\_count; i++) {

printf("%d", dominant[i]);

if (i < dominant\_count - 1) printf(", ");

}

printf("]\n");

printf("\n=== ЗАДАНИЕ 2 ===\n\n");

int max\_edges = N \* (N - 1) / 2 + N;

int edge\_count = 0;

int loop\_count = 0;

// Находим все ребра и петли

int\*\* edges\_list = (int\*\*)malloc(max\_edges \* sizeof(int\*));

for (int i = 0; i < max\_edges; i++) {

edges\_list[i] = (int\*)malloc(2 \* sizeof(int));

}

// Сначала добавляем петли

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

if (adj\_matrix[i][i] == 1) {

edges\_list[edge\_count][0] = i;

edges\_list[edge\_count][1] = i; // петля

edge\_count++;

loop\_count++;

}

}

// Затем добавляем обычные ребра

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

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

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

edges\_list[edge\_count][0] = i;

edges\_list[edge\_count][1] = j;

edge\_count++;

}

}

}

// Создаем матрицу инцидентности

int\*\* inc\_matrix = (int\*\*)malloc(N \* sizeof(int\*));

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

inc\_matrix[i] = (int\*)malloc((edge\_count > 0 ? edge\_count : 1) \* sizeof(int));

}

// Заполняем матрицу инцидентности нулями

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

for (int j = 0; j < edge\_count; j++) {

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

}

}

// Заполняем единицами (для петель ставим 2 в одной ячейке)

for (int j = 0; j < edge\_count; j++) {

int v1 = edges\_list[j][0];

int v2 = edges\_list[j][1];

if (v1 == v2) {

// Петля - ставим 2 в соответствующей вершине (ПЕТЛЯ ДАЕТ +2)

inc\_matrix[v1][j] = 2;

}

else {

// Обычное ребро

inc\_matrix[v1][j] = 1;

inc\_matrix[v2][j] = 1;

}

}

printf("1. Матрица инцидентности:\n");

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

for (int j = 0; j < edge\_count; j++) {

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

}

printf("\n");

}

if (edge\_count > 0) {

printf("Ребра: [");

for (int i = 0; i < edge\_count; i++) {

if (edges\_list[i][0] == edges\_list[i][1]) {

printf("(%d) петля", edges\_list[i][0]); // петля

}

else {

printf("(%d,%d)", edges\_list[i][0], edges\_list[i][1]); // обычное ребро

}

if (i < edge\_count - 1) printf(", ");

}

printf("]\n");

}

else {

printf("Ребра: []\n");

}

// 2. Размер графа

printf("\n2. Размер графа: %d ребер (из них %d петель)\n", edge\_count, loop\_count);

int\* isolated\_inc = (int\*)malloc(N \* sizeof(int));

int\* pendant\_inc = (int\*)malloc(N \* sizeof(int));

int\* dominant\_inc = (int\*)malloc(N \* sizeof(int));

int isolated\_count\_inc = 0, pendant\_count\_inc = 0, dominant\_count\_inc = 0;

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

int degree = 0;

int degree\_without\_loops = 0;

for (int j = 0; j < edge\_count; j++) {

if (inc\_matrix[i][j] == 2) {

degree += 2;

}

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

degree += 1;

degree\_without\_loops += 1;

}

}

if (degree\_without\_loops == 0) {

isolated\_inc[isolated\_count\_inc] = i;

isolated\_count\_inc++;

}

else if (degree\_without\_loops == 1) {

pendant\_inc[pendant\_count\_inc] = i;

pendant\_count\_inc++;

}

if (degree >= N) {

dominant\_inc[dominant\_count\_inc] = i;

dominant\_count\_inc++;

}

}

printf("\n3. Особые вершины:\n");

printf(" Изолированные: [");

for (int i = 0; i < isolated\_count\_inc; i++) {

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

if (i < isolated\_count\_inc - 1) printf(", ");

}

printf("]\n");

printf(" Концевые: [");

for (int i = 0; i < pendant\_count\_inc; i++) {

printf("%d", pendant\_inc[i]);

if (i < pendant\_count\_inc - 1) printf(", ");

}

printf("]\n");

printf(" Доминирующие: [");

for (int i = 0; i < dominant\_count\_inc; i++) {

printf("%d", dominant\_inc[i]);

if (i < dominant\_count\_inc - 1) printf(", ");

}

printf("]\n");

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

free(adj\_matrix[i]);

free(inc\_matrix[i]);

}

free(adj\_matrix);

free(inc\_matrix);

for (int i = 0; i < max\_edges; i++) {

free(edges\_list[i]);

}

free(edges\_list);

free(isolated);

free(pendant);

free(dominant);

free(isolated\_inc);

free(pendant\_inc);

free(dominant\_inc);

return 0;

}