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»

Выполнил: ст. гр. 24ВВВ1

Гурин А.Н.

Захаров А.В.

Принял:

к.т.н, доцент Юрова О. В.

к.т.н. Деев М.В.

Пенза 2025

**Цель работы:** сгенерировать матрицы смежности и инцидентности неориентированного графа и определить его основные характеристики.

**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.

**Теоретическая часть:**

Матрица смежности неориентированного графа – квадратная матрица *A(G) = [aij]*, порядка *n*, у которой *aij = { 1, если (vi, vj) принадлежит X, 0, если (vi, vj) не принадлежит X}*.

Матрица инцидентности неориентированного графа – матрица размера *n x m B(G) = [bij]*, у которой *bij = { 1, если vi инцидентно ребру xj , 0, если vi не инцидентно ребру xj}*.

Размер графа – кардинальное число множества ребер или, другими словами, количество ребер

Изолированные вершины неориентированного графа – вершина, которая не является концом ни одного ребра.

Концевые вершины неориентированного графа – вершина, которая является концом ровно одного ребра.

Доминирующие вершины неориентированного графа – вершина графа смежная с каждой дугой.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n;

int chance\_rebra;

int chance\_petli;

printf("Введите количество вершин графа: ");

scanf("%d", &n);

printf("Введите вероятность ребра между вершинами (0-100%%): ");

scanf("%d", &chance\_rebra);

if (chance\_rebra > 100) {

printf("Максимальная вероятность 100%%. Выставлено 100%%.\n");

chance\_rebra = 100;

}

else if (chance\_rebra < 0) {

printf("Минимальная вероятность 0%%. Выставлено 0%%.\n");

chance\_rebra = 0;

}

printf("Введите вероятность петли у вершины (0-100%%): ");

scanf("%d", &chance\_petli);

if (chance\_petli > 100) {

printf("Максимальная вероятность 100%%. Выставлено 100%%.\n");

chance\_petli = 100;

}

else if (chance\_petli < 0) {

printf("Минимальная вероятность 0%%. Выставлено 0%%.\n");

chance\_petli = 0;

}

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

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

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

}

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

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

if (i == j) {

matrix[i][j] = (rand() % 100 < chance\_petli) ? 1 : 0;

}

else {

matrix[i][j] = (rand() % 100 < chance\_rebra) ? 1 : 0;

matrix[j][i] = matrix[i][j];

}

}

}

printf("\nМатрица смежности:\n");

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

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

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

}

printf("\n");

}

int rebra = 0;

int petli = 0;

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

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

if (i == j) {

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

rebra++;

petli++;

}

}

else {

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

rebra++;

}

}

}

}

printf("\nСтатистика графа:\n");

printf("Всего ребер: %d\n", rebra);

if (petli > 0) {

printf("Из них петель: %d\n", petli);

}

if (rebra % 10 == 1 && rebra % 100 != 11) {

printf("Размер графа: %d ребро\n", rebra);

}

else if (rebra % 10 >= 2 && rebra % 10 <= 4 && (rebra % 100 < 10 || rebra % 100 > 20)) {

printf("Размер графа: %d ребра\n", rebra);

}

else {

printf("Размер графа: %d рёбер\n", rebra);

}

printf("\nАнализ вершин по матрице смежности:\n");

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

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

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

int list\_count = 0, izol\_count = 0, dom\_count = 0;

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

int has\_connections = 0;

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

if (i != j && matrix[i][j] == 1) {

has\_connections = 1;

break;

}

}

int has\_loop = (matrix[i][i] == 1);

int stepen = 0;

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

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

if (i == j) stepen += 2;

else stepen += 1;

}

}

if (stepen == 1 && !has\_loop) {

list[list\_count] = i + 1;

list\_count++;

}

if (!has\_connections) {

izol[izol\_count] = i + 1;

izol\_count++;

}

int all\_connected = 1;

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

if (i != j && matrix[i][j] != 1) {

all\_connected = 0;

break;

}

}

if (all\_connected) {

dom[dom\_count] = i + 1;

dom\_count++;

}

}

printf("Концевые вершины (%d): ", list\_count);

if (list\_count == 0) printf("нет");

else for (int i = 0; i < list\_count; i++) printf("%d ", list[i]);

printf("\nИзолированные вершины (%d): ", izol\_count);

if (izol\_count == 0) printf("нет");

else for (int i = 0; i < izol\_count; i++) printf("%d ", izol[i]);

printf("\nДоминирующие вершины (%d): ", dom\_count);

if (izol\_count > 0) printf("не могут быть при наличии изолированных");

else if (dom\_count == 0) printf("нет");

else for (int i = 0; i < dom\_count; i++) printf("%d ", dom[i]);

int vsego\_reber = 0;

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

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

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

vsego\_reber++;

}

}

}

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

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

incidency[i] = (int\*)malloc(vsego\_reber \* sizeof(int));

for (int j = 0; j < vsego\_reber; j++) {

incidency[i][j] = 0;

}

}

int index\_rebra = 0;

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

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

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

if (i == j) {

incidency[i][index\_rebra] = 2;

}

else {

incidency[i][index\_rebra] = 1;

incidency[j][index\_rebra] = 1;

}

index\_rebra++;

}

}

}

printf("\n\nМатрица инцидентности:\n");

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

for (int j = 0; j < vsego\_reber; j++) {

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

}

printf("\n");

}

printf("\nАнализ вершин по матрице инцидентности:\n");

int\* list\_inc = (int\*)malloc(n \* sizeof(int));

int\* izol\_inc = (int\*)malloc(n \* sizeof(int));

int\* dom\_inc = (int\*)malloc(n \* sizeof(int));

int list\_count\_inc = 0, izol\_count\_inc = 0, dom\_count\_inc = 0;

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

int has\_connections\_inc = 0;

for (int j = 0; j < vsego\_reber; j++) {

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

has\_connections\_inc = 1;

break;

}

}

int stepen = 0;

for (int j = 0; j < vsego\_reber; j++) {

if (incidency[i][j] == 1) stepen += 1;

else if (incidency[i][j] == 2) stepen += 2;

}

if (stepen == 1 && incidency[i][i] != 2) {

list\_inc[list\_count\_inc] = i + 1;

list\_count\_inc++;

}

if (!has\_connections\_inc) {

izol\_inc[izol\_count\_inc] = i + 1;

izol\_count\_inc++;

}

int all\_connected\_inc = 1;

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

if (i != k) {

int connected = 0;

for (int j = 0; j < vsego\_reber; j++) {

if (incidency[i][j] == 1 && incidency[k][j] == 1) {

connected = 1;

break;

}

}

if (!connected) {

all\_connected\_inc = 0;

break;

}

}

}

if (all\_connected\_inc) {

dom\_inc[dom\_count\_inc] = i + 1;

dom\_count\_inc++;

}

}

printf("Концевые вершины (%d): ", list\_count\_inc);

if (list\_count\_inc == 0) printf("нет");

else for (int i = 0; i < list\_count\_inc; i++) printf("%d ", list\_inc[i]);

printf("\nИзолированные вершины (%d): ", izol\_count\_inc);

if (izol\_count\_inc == 0) printf("нет");

else for (int i = 0; i < izol\_count\_inc; i++) printf("%d ", izol\_inc[i]);

printf("\nДоминирующие вершины (%d): ", dom\_count\_inc);

if (izol\_count\_inc > 0) printf("не могут быть при наличии изолированных");

else if (dom\_count\_inc == 0) printf("нет");

else for (int i = 0; i < dom\_count\_inc; i++) printf("%d ", dom\_inc[i]);

printf("\n");

free(list);

free(izol);

free(dom);

free(list\_inc);

free(izol\_inc);

free(dom\_inc);

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

free(matrix[i]);

free(incidency[i]);

}

free(matrix);

free(incidency);

return 0;

}

**Практическая часть:**

**Задание 1**

Для решения первой задачи был реализован алгоритм генерации случайного неориентированного графа с использованием генератора псевдослучайных чисел, инициализируемого текущим временем системы, что обеспечивает различный результат при каждом запуске программы. Пользователь имеет возможность задать вероятность образования ребер между различными вершинами и вероятность образования петель у вершин, что позволяет гибко настраивать плотность графа. Матрица смежности формируется построчно с учетом свойства симметричности неориентированного графа - для элементов главной диагонали (i = j) вероятность установки значения 1 определяется параметром chance\_petli, а для недиагональных элементов (i ≠ j) - параметром chance\_rebra, при этом симметричность обеспечивается одновременным заполнением элементов matrix[i][j] и matrix[j][i]. Определение размера графа выполняется путем анализа верхней треугольной части матрицы смежности включительно с главной диагональю, где алгоритм последовательно проверяет все элементы для i ≤ j: при i = j и matrix[i][j] = 1 фиксируется петля и увеличивается счетчик ребер, а при i ≠ j и matrix[i][j] = 1 фиксируется обычное ребро, такой подход исключает двойной учет неориентированных ребер и корректно учитывает петли как полноценные ребра. Классификация специальных вершин выполняется на основе вычисления степеней вершин с учетом особенностей петель: степень вершины вычисляется как сумма элементов строки матрицы смежности с учетом того, что петля (элемент matrix[i][i] = 1) увеличивает степень на 2, концевые вершины определяются как вершины со степенью 1 при отсутствии петель, изолированные вершины идентифицируются по отсутствию связей с другими вершинами (все недиагональные элементы строки равны 0), а доминирующие вершины определяются как вершины, связанные со всеми остальными вершинами графа.

**Задание 2**

Построение матрицы инцидентности выполняется после определения общего количества ребер из матрицы смежности, где алгоритм создает матрицу размером n×m, где n - количество вершин, m - общее количество ребер включая петли. Для каждого ребра, обнаруженного в матрице смежности: обычное ребро (i < j и matrix[i][j] = 1) представляется столбцом с двумя единицами в строках i и j, а петля (i = j и matrix[i][j] = 1) представляется столбцом с одним ненулевым элементом со значением 2 в строке i. Определение размера графа через матрицу инцидентности тривиально и равно количеству столбцов созданной матрицы, что соответствует общему количеству ребер, ранее вычисленному из матрицы смежности. Анализ специальных вершин через матрицу инцидентности основан на концепции регулярной степени: регулярная степень вершины вычисляется как количество столбцов, в которых для данной вершины установлено значение 1 (только обычные ребра), концевые вершины определяются по регулярной степени равной 1 при отсутствии петель, изолированные вершины идентифицируются по нулевой регулярной степени и отсутствию петель, а доминирующие вершины определяются через проверку наличия связей со всеми остальными вершинами путем анализа попарной инцидентности в матрице инцидентности - для каждой пары вершин проверяется существование столбца, где обе вершины имеют значение 1. Особенности реализации включают обеспечение согласованности результатов между двумя представлениями графа, корректную обработку петель в обоих представлениях, проверку входных данных на валидность, корректное управление памятью с освобождением всех выделенных ресурсов и лингвистически правильную обработку числительных в выводе статистики, что делает программу надежным инструментом для анализа свойств графов.

**Вывод:**

В ходе выполнения данной лабораторной работы была успешно разработана и реализована комплексная система анализа неориентированных графов, поддерживающая два различных матричных представления - смежности и инцидентности. Практическая значимость работы подтверждена полной функциональной готовностью всех алгоритмов и их взаимной согласованностью. Особенно важной оказалась корректная обработка петель, которая была реализована через систему дополнительных массивов маркеров, что обеспечило точность вычисления степеней вершин в обоих представлениях.