Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Отчёт

по лабораторной работе №5

по курсу «Л и О А в ИЗ»

на тему «Определение характеристик графов»

Выполнили студенты группы 24ВВВ4:

Кошелев Р.Д.

Кондратьев С.В.

Приняли к.т.н., доцент:

Юрова О.В.

к.э.н., доцент:

Акифьев И.В.

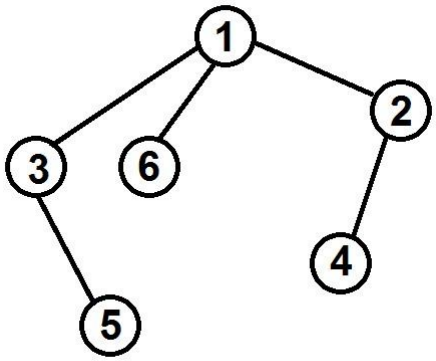
Пенза 2025

Цель: освоить методы представления графов с использованием матрицы смежности и разработать алгоритмы анализа структурных свойств графов. Научиться генерировать матрицы смежности для неориентированных графов, определять размер графа и находить специальные типы вершин. Получить практические навыки работы с двумерными массивами в языке программирования С.

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 $deg(v_i)$.



$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

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**.

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

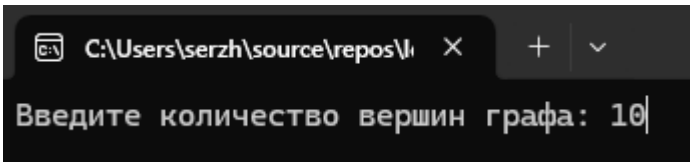


Рисунок 1 – Ввод количества вершин

```
Матрица смежности графа G:
0 0 1 1 1 1 0 0 1 0
0 1 1 0 0 0 0 1 1 0
1 1 1 1 0 1 0 1 1 1
1 0 1 0 1 1 0 1 0 0
1 0 0 1 0 0 1 1 1 0
1 0 1 1 0 0 1 0 0 1
0 0 0 0 1 1 0 1 1 0
0 1 1 1 1 0 1 1 0 1
1 1 1 0 1 0 1 0 0 0
0 0 1 0 0 1 0 1 0 1

--- Размер графа G ---
Сумма всех элементов матрицы: 52
Количество ребер (размер графа): 28

--- Особые вершины графа G (через матрицу смежности) ---
Вершина 0: степень = 5
Вершина 1: степень = 4
Вершина 2: степень = 8
Вершина 3: степень = 5
Вершина 4: степень = 5
Вершина 5: степень = 5
Вершина 6: степень = 4
Вершина 7: степень = 7
Вершина 8: степень = 5
Вершина 9: степень = 4
```

Рисунок 2 – Вывод матрицы смежности, размера графа, количества ребер, степеней вершин

```
--- Результаты (матрица смежности) ---
Изолированных вершин нет
Концевых вершин нет
Доминирующих вершин нет

--- Матрица инцидентности ---
Общее количество рёбер: 28

Матрица инцидентности (10 вершин * 28 рёбер):
    e0  e1  e2  e3  e4  e5  e6  e7  e8  e9  e10 e11 e12 e13 e14 e15 e16 e17 e18 e19 e20 e21 e22 e23 e24 e25 e26 e27
v0  1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
v1  0   0   0   0   0   2   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
v2  1   0   0   0   0   0   1   0   0   2   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0
v3  0   1   0   0   0   0   0   0   0   0   1   0   0   0   0   1   1   1   0   0   0   0   0   0   0   0   0   0
v4  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   1   1   0   0   0   0   0   0   0
v5  0   0   0   1   0   0   0   0   0   0   0   1   0   0   0   0   1   0   0   0   0   1   1   0   0   0   0   0
v6  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   0   1   1   0   0   0
v7  0   0   0   0   0   0   0   1   0   0   0   0   1   0   0   0   0   1   0   1   0   0   0   1   0   2   1   0
v8  0   0   0   0   1   0   0   0   1   0   0   0   0   1   0   0   0   0   0   0   1   0   0   0   1   0   0   0
v9  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   1   0   0   0   1   2
```

Рисунок 3 – Результаты матрицы смежности, вывод матрицы инцидентности

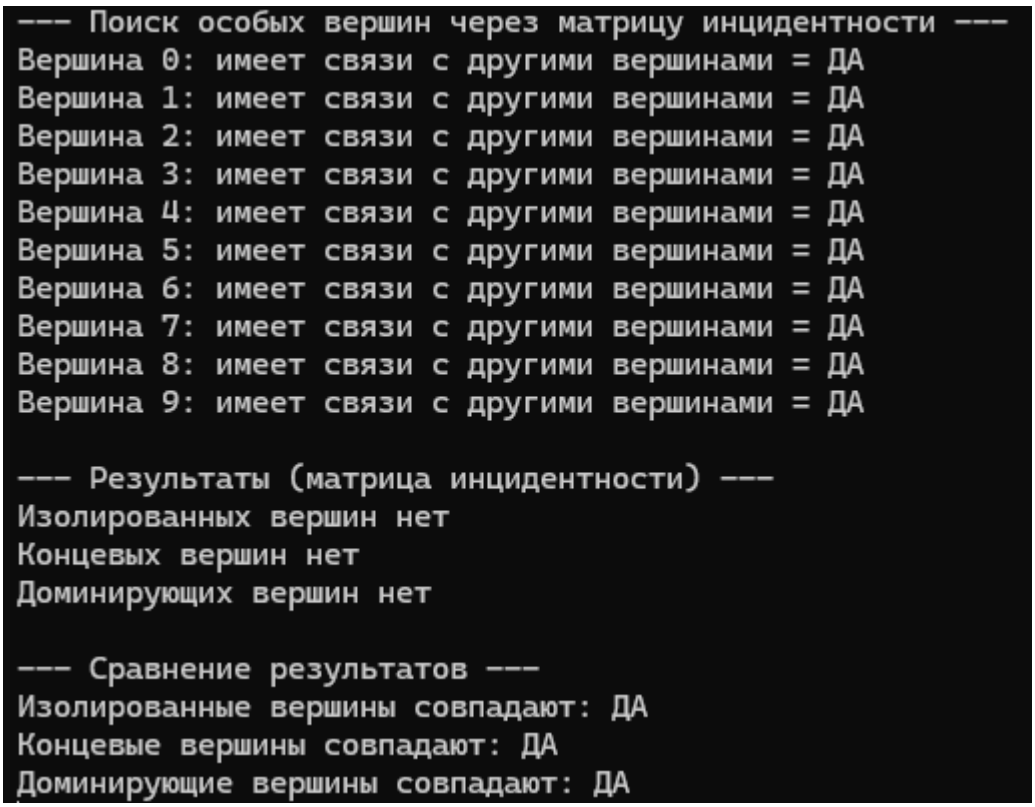


Рисунок 4 – Результаты матрицы инцидентности, сравнение результатов, проверка связи

Вывод: освоили методы представления графов с использованием матрицы смежности и разработали алгоритмы анализа структурных свойств графов. Научились генерировать матрицы смежности для неориентированных графов, определять размер графа и находить специальные типы вершин. Получили практические навыки работы с двумерными массивами в языке программирования C.

Приложение А

Листинг

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>
#include <conio.h>

int main() {
    setlocale(LC_ALL, "Rus");
    int n;

    printf("Введите количество вершин графа: ");
    scanf("%d", &n);

    int** matrix = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        matrix[i] = (int*)malloc(n * sizeof(int));
    }

    int* isolated = (int*)malloc(n * sizeof(int));
    int* terminal = (int*)malloc(n * sizeof(int));
    int* dominant = (int*)malloc(n * sizeof(int));

    srand(time(NULL));
```

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        if (i == j) {
            matrix[i][j] = rand() % 2;  // петля
        }
        else {
            int oneORzero = rand() % 2;
            matrix[i][j] = oneORzero;
            matrix[j][i] = oneORzero;
        }
    }
}

printf("\nМатрица смежности графа G:\n");
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        printf("%d ", matrix[i][j]);
    }
    printf("\n");
}

// ЗАДАНИЕ 2: Определение размера графа через матрицу смежности
int sum = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        sum += matrix[i][j];
    }
```

```
}

// Размер графа = (общая сумма - сумма петель) / 2 + сумма петель
int loop_sum = 0;
for (int i = 0; i < n; i++) {
    loop_sum += matrix[i][i];
}
int graph_size = (sum - loop_sum) / 2 + loop_sum;

printf("\n--- Размер графа G ---\n");
printf("Сумма всех элементов матрицы: %d\n", sum);
printf("Количество ребер (размер графа): %d\n", graph_size);

// ЗАДАНИЕ 3: Поиск особых вершин через матрицу смежности
printf("\n--- Особые вершины графа G (через матрицу смежности) ---\n");

int iso_count = 0, term_count = 0, dom_count = 0;

// Анализ вершин
for (int i = 0; i < n; i++) {
    int degree = 0;

    for (int j = 0; j < n; j++) {
        if (i == j) {
            degree += matrix[i][j];  // петля дает степень 1
        }
        else {
```

```
            degree += matrix[i][j];  // обычное ребро
        }
    }

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

    if (degree == 0) {
        isolated[iso_count++] = i;
    }
    else if (degree == 1) {
        // Проверяем, что это действительно концевая вершина (не петля)
        int has_loop = (matrix[i][i] > 0);
        int normal_edges = 0;
        for (int j = 0; j < n; j++) {
            if (i != j && matrix[i][j] > 0) {
                normal_edges++;
            }
        }

        if (!has_loop && normal_edges == 1) {
            terminal[term_count++] = i;
        }
    }
}

// Поиск доминирующих вершин
for (int i = 0; i < n; i++) {
```

```
    int is_dominant = 1;
    for (int j = 0; j < n; j++) {
        if (i != j && matrix[i][j] == 0) {
            is_dominant = 0;
            break;
        }
    }
    if (is_dominant) {
        dominant[dom_count++] = i;
    }
}

// Вывод результатов для матрицы смежности
printf("\n--- Результаты (матрица смежности) ---\n");

if (iso_count > 0) {
    printf("Изолированные вершины: ");
    for (int i = 0; i < iso_count; i++) printf("%d ", isolated[i]);
    printf("\n");
}
else {
    printf("Изолированных вершин нет\n");
}

if (term_count > 0) {
    printf("Концевые вершины: ");
    for (int i = 0; i < term_count; i++) printf("%d ", terminal[i]);
```

```
    printf("\n");
}
else {
    printf("Концевых вершин нет\n");
}

if (dom_count > 0) {
    printf("Доминирующие вершины: ");
    for (int i = 0; i < dom_count; i++) printf("%d ", dominant[i]);
    printf("\n");
}
else {
    printf("Доминирующих вершин нет\n");
}

// ЗАДАНИЕ 4: Построение матрицы инцидентности
printf("\n--- Матрица инцидентности ---\n");

// Подсчёт количества рёбер для матрицы инцидентности
int edge_count = 0;
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        if (matrix[i][j] > 0) {
            if (i == j) {
                edge_count += matrix[i][j];  // петли
            }
            else {
```

```
                edge_count += matrix[i][j];  // обычные рёбра
            }
        }
    }
}

printf("Общее количество рёбер: %d\n", edge_count);

// Создание матрицы инцидентности
int** incidence = NULL;
if (edge_count > 0) {
    incidence = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        incidence[i] = (int*)calloc(edge_count, sizeof(int));
    }

    // Заполнение матрицы инцидентности
    int edge_index = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            if (matrix[i][j] > 0) {
                if (i == j) {
                    // Петля
                    for (int k = 0; k < matrix[i][j]; k++) {
                        incidence[i][edge_index] = 2;
                        edge_index++;
                    }
```

```
            }
            else {
                // Обычное ребро
                for (int k = 0; k < matrix[i][j]; k++) {
                    incidence[i][edge_index] = 1;
                    incidence[j][edge_index] = 1;
                    edge_index++;
                }
            }
        }
    }
}

// Вывод матрицы инцидентности
printf("\nМатрица инцидентности (%d вершин * %d рёбер):\n", n, edge_count);
printf("    ");
for (int j = 0; j < edge_count; j++) {
    printf("e%-2d ", j);
}
printf("\n");

for (int i = 0; i < n; i++) {
    printf("v%-2d", i);
    for (int j = 0; j < edge_count; j++) {
        printf("%-3d ", incidence[i][j]);
    }
```

```
            printf("\n");
        }
    }
    else {
        printf("В графе нет рёбер, матрица инцидентности пуста\n");
    }

    // ЗАДАНИЕ 5: Поиск особых вершин через матрицу инцидентности (ИСПРАВЛЕННАЯ ЛОГИКА)
        // ЗАДАНИЕ 5: Поиск особых вершин через матрицу инцидентности (ИСПРАВЛЕННАЯ ЛОГИКА)
    printf("\n--- Поиск особых вершин через матрицу инцидентности ---\n");

    int iso_count_inc = 0, term_count_inc = 0, dom_count_inc = 0;
    int* isolated_inc = (int*)malloc(n * sizeof(int));
    int* terminal_inc = (int*)malloc(n * sizeof(int));
    int* dominant_inc = (int*)malloc(n * sizeof(int));

    if (edge_count > 0) {
        // Анализ вершин через матрицу инцидентности
        for (int i = 0; i < n; i++) {
            int has_connections_to_other_vertices = 0;  // Флаг связей с другими вершинами
            int has_only_loops = 1;  // Предполагаем, что есть только петли

            for (int j = 0; j < edge_count; j++) {
                if (incidence[i][j] == 1) {
                    // Обычное ребро - проверяем, соединяет ли с другой вершиной
```

```
            for (int k = 0; k < n; k++) {
                if (k != i && incidence[k][j] == 1) {
                    has_connections_to_other_vertices = 1;
                    has_only_loops = 0;
                    break;
                }
            }
        }
    }

    printf("Вершина %d: имеет связи с другими вершинами = %s\n",
        i, has_connections_to_other_vertices ? "ДА" : "НЕТ");
    if (!has_connections_to_other_vertices) {
        isolated_inc[iso_count_inc++] = i;
    }
}

// Дополнительный анализ для концевых вершин
for (int i = 0; i < n; i++) {
    int connections_count = 0;  // Количество связей с другими вершинами

    for (int j = 0; j < edge_count; j++) {
        if (incidence[i][j] == 1) {
            // Обычное ребро - считаем связь с другой вершиной
            for (int k = 0; k < n; k++) {
                if (k != i && incidence[k][j] == 1) {
                    connections_count++;
```

```
                break;
            }
        }
    }
}

    // Концевая вершина (ровно одна связь с другой вершиной)
    if (connections_count == 1) {
        terminal_inc[term_count_inc++] = i;
    }
}

// Поиск доминирующих вершин через матрицу инцидентности
for (int i = 0; i < n; i++) {
    int is_dominant = 1;

    // Проверяем все другие вершины
    for (int j = 0; j < n; j++) {
        if (i != j) {
            int connected = 0;
            // Ищем ребро между вершинами i и j
            for (int k = 0; k < edge_count; k++) {
                if (incidence[i][k] == 1 && incidence[j][k] == 1) {
                    connected = 1;
                    break;
                }
            }
```

```
            if (!connected) {
                is_dominant = 0;
                break;
            }
        }
    }

    if (is_dominant) {
        dominant_inc[dom_count_inc++] = i;
    }
  }
}
else {
  // Если нет рёбер, все вершины изолированные
  for (int i = 0; i < n; i++) {
    isolated_inc[iso_count_inc++] = i;
  }
}

// Вывод результатов для матрицы инцидентности
printf("\n--- Результаты (матрица инцидентности) ---\n");

if (iso_count_inc > 0) {
  printf("Изолированные вершины: ");
  for (int i = 0; i < iso_count_inc; i++) printf("%d ", isolated_inc[i]);
  printf("\n");
}
```

```
else {
    printf("Изолированных вершин нет\n");
}

if (term_count_inc > 0) {
    printf("Концевые вершины: ");
    for (int i = 0; i < term_count_inc; i++) printf("%d ", terminal_inc[i]);
    printf("\n");
}
else {
    printf("Концевых вершин нет\n");
}

if (dom_count_inc > 0) {
    printf("Доминирующие вершины: ");
    for (int i = 0; i < dom_count_inc; i++) printf("%d ", dominant_inc[i]);
    printf("\n");
}
else {
    printf("Доминирующих вершин нет\n");
}

// Сравнение результатов двух методов
printf("\n--- Сравнение результатов ---\n");

// Сравнение изолированных вершин
int iso_match = 1;
```

```
    if (iso_count != iso_count_inc) {
        iso_match = 0;
    }
    else {
        for (int i = 0; i < iso_count; i++) {
            if (isolated[i] != isolated_inc[i]) {
                iso_match = 0;
                break;
            }
        }
    }
    printf("Изолированные вершины совпадают: %s\n", iso_match ? "ДА" :
"НЕТ");


    // Сравнение концевых вершин
    int term_match = 1;
    if (term_count != term_count_inc) {
        term_match = 0;
    }
    else {
        for (int i = 0; i < term_count; i++) {
            if (terminal[i] != terminal_inc[i]) {
                term_match = 0;
                break;
            }
        }
    }
```

```
    printf("Концевые вершины совпадают: %s\n", term_match ? "ДА" : "НЕТ");

    // Сравнение доминирующих вершин
    int dom_match = 1;
    if (dom_count != dom_count_inc) {
        dom_match = 0;
    }
    else {
        for (int i = 0; i < dom_count; i++) {
            if (dominant[i] != dominant_inc[i]) {
                dom_match = 0;
                break;
            }
        }
    }
    printf("Доминирующие вершины совпадают: %s\n", dom_match ? "ДА" :
"НЕТ");

    // Очистка памяти
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);

    if (edge_count > 0) {
        for (int i = 0; i < n; i++) {
            free(incidence[i]);
```

```
        }
        free(incidence);
    }

    free(isolated);
    free(terminal);
    free(dominant);
    free(isolated_inc);
    free(terminal_inc);
    free(dominant_inc);

    _getch();
    return 0;
}
```