

Министерство науки и высшего образования РФ  
Пензенский государственный университет  
Кафедра “Вычислительная техника”

## **Отчёт**

по лабораторной работе №5  
по курсу “Логика и основы алгоритмизации в инженерных задачах”  
на тему “Определение характеристик графа”

Выполнил студент гр. 22ВВВЗ:  
Кулахметов С.И.

Приняли:  
к.т.н., доцент Юрова О.В.  
к.э.н., доцент Акифьев И.В.

Пенза 2023

## Цель работы

Ознакомиться с понятием «граф». Научиться реализовывать его при помощи матрицы смежности и матрицы инцидентности. Выполнить лабораторное задание.

## Лабораторное задание

### Задание 1

1. Сгенерировать (используя генератор случайных чисел) матрицу смежности для неориентированного графа  $G$ . Вывести матрицу на экран.

2. Определить размер графа  $G$ , используя матрицу смежности графа.

3. Найти изолированные, концевые и доминирующие вершины.

### Задание 2

1. Построить для графа  $G$  матрицу инцидентности.

2. Определить размер графа  $G$ , используя матрицу инцидентности.

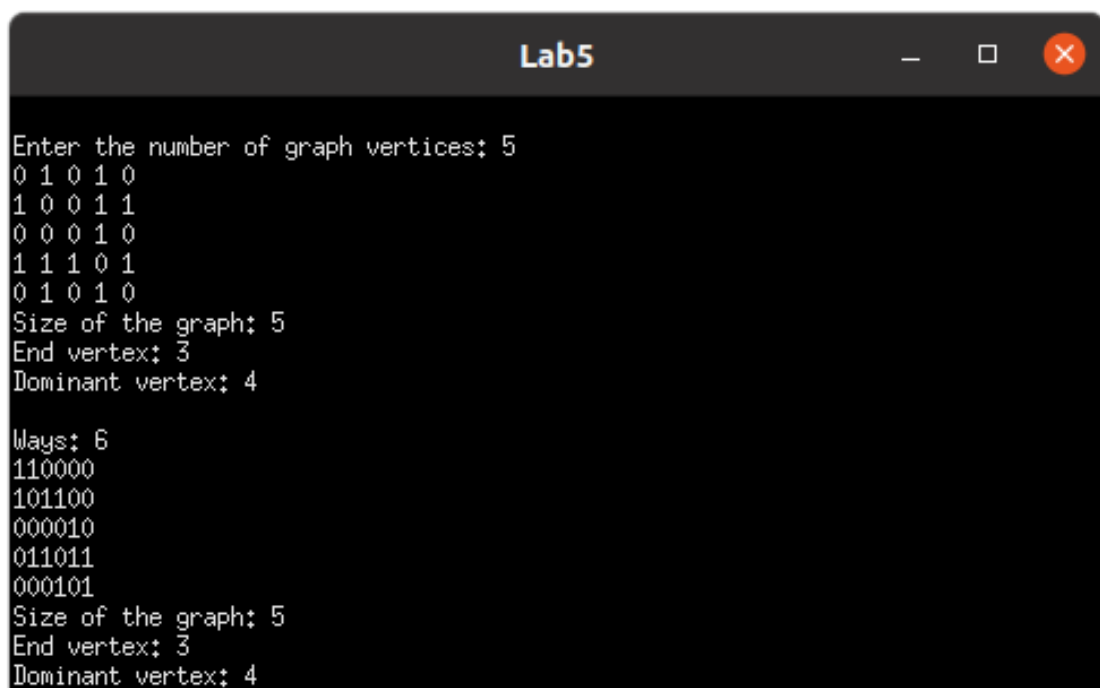
3. Найти изолированные, концевые и доминирующие вершины.

## Пояснительный текст к программе

В программе инициализированы два основных двумерных массива: `graph[m][n]` — отвечает за хранение матрицы смежности неориентированного графа; `Graph[m][k]` — отвечает за хранение матрицы инцидентности, построенной на основе данных предыдущего массива. Где  $m$  и  $n$  — количество вершин графа,  $k$  — количество рёбер.

## Результаты работы программы

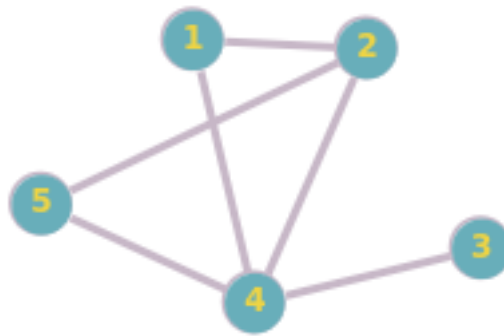
Построение матриц смежности и инцидентности.



```
Lab5
Enter the number of graph vertices: 5
0 1 0 1 0
1 0 0 1 1
0 0 0 1 0
1 1 1 0 1
0 1 0 1 0
Size of the graph: 5
End vertex: 3
Dominant vertex: 4

Ways: 6
110000
101100
000010
011011
000101
Size of the graph: 5
End vertex: 3
Dominant vertex: 4
```

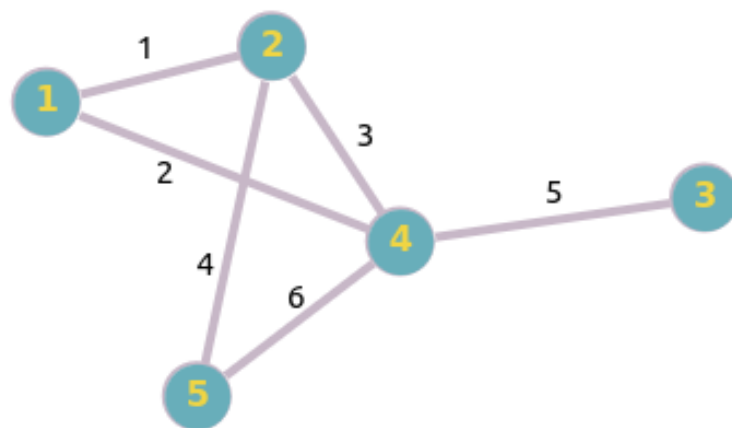
Граф, сгенерированный по матрице смежности.



Вершина 3 — концевая.

Вершина 4 — доминирующая.

Граф, сгенерированный по матрице инцидентности.



Вершина 3 — концевая.

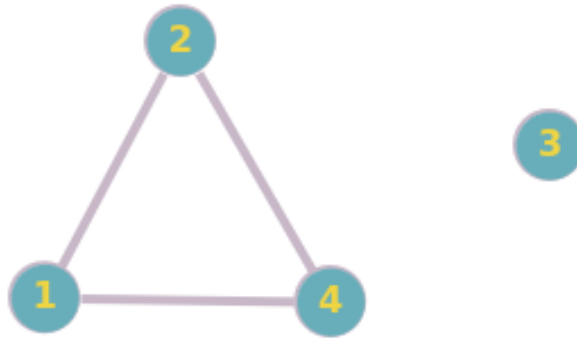
Вершина 4 — доминирующая.

Построение матриц смежности и инцидентности.

```
Lab5
# Graphs #
Enter the number of graph vertices: 4
0 1 0 1
1 0 0 1
0 0 0 0
1 1 0 0
Size of the graph: 4
Isolated vertex: 3

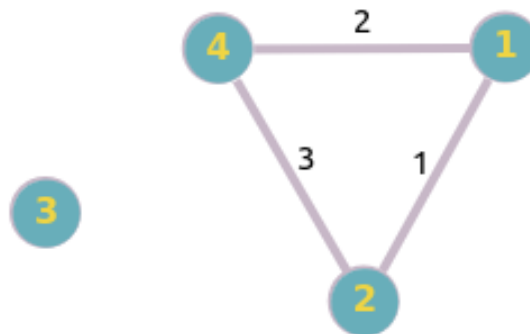
Ways: 3
110
101
000
011
Size of the graph: 4
Isolated vertex: 3
```

Граф, сгенерированный по матрице смежности.



Вершина 3 — изолированная.

Граф, сгенерированный по матрице инцидентности.



Вершина 3 — изолированная.

### Вывод

В ходе выполнения данной лабораторной работы были получены навыки представления неориентированных графов при помощи матриц смежности и инцидентности; нахождения концевых, изолированных и доминирующих вершин неориентированного графа.

**Ссылка на *GitHub* репозиторий с лабораторной работой**

<https://github.com/KulakhmetovS/Lab5>

## Приложение А

### Листинг программы

#### Файл main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    srand(time(NULL));

    int size, count = 0, i, j, graph_size = 0, ways = 0;
    int **graph;

    printf("\t# Graphs #\n\n");
    printf("Enter the number of graph vertices: ");
    scanf("%d", &size);

    // <----- Task 1. Creating the graph ----->
    // Memory allocation
    graph = (int **)(malloc(sizeof(int *) * (size + 1)));
    for(i = 0; i < size; i++)
        graph[i] = (int *)(malloc(sizeof(int *) * size));

    graph[i + 1] = malloc(sizeof(int));
    graph[i + 1] = NULL;

    // Filling the matrix
    for(i = 0; i < size; i++)
        for(j = i; j < size; j++)
        {
            graph[i][j] = rand() % 2;
            graph[j][i] = graph[i][j];
            if(i == j) graph[i][j] = 0;
            if(graph[i][j] == 1) ways++;
        }

    // Printing the matrix
    for(i = 0; i < size; i++)
    {
        for(j = 0; j < size; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }

    // Finding size of the graph
    int **pointer = graph; // Saving original pointer
    while(*graph != NULL)
    {
        graph++;
        graph_size++;
    }
    graph = pointer;

    printf("Size of the graph: %d\n", graph_size);

    // Finding isolated vertices
```

```

for(i = 0; i < size; i++)
{
    count = 0;
    for(j = 0; j < size; j++)
        if(graph[i][j] == 0) count++;

    if(count == size) printf("Isolated vertex: %d\n", i+1);
}

// Finding end vertices
for(i = 0; i < size; i++)
{
    count = 0;
    for(j = 0; j < size; j++)
    {
        if(graph[i][j] == 1) count++;
    }
    if(count == 1) printf("End vertex: %d\n", i+1);
    else if(count == (size - 1)) printf("Dominant vertex: %d\n", i+1);
}

// <----- Task 2. Creating the graph ----->

//int **Graph;
int Graph[size][ways];
int **vert;
int k = 0;

vert = (int **)malloc(sizeof(int *) * ways); // array for vertices
for(i = 0; i < ways; i++)
    vert[i] = (int *)malloc(sizeof(int *) * 2);

/*Graph = (int **)(malloc(sizeof(int *) * ways));
for(i = 0; i < ways; i++)
    Graph[i] = (int *)malloc(sizeof(int *) * size);*/

for(i = 0; i < size; i++)
{
    for(j = 0; j < ways; j++)
    {
        Graph[i][j] = 0;
    }
}

printf("\nWays: %d\n", ways);

    for(i = 0; i < size; i++)
    {
        for(j = i; j < size; j++)
        {
            if(graph[i][j] == 1)
            {
                if(ways > 1)
                {
                    vert[0][k] = i;
                    vert[1][k] = j;
                    k++;
                }
            }
            else if(ways == 1)

```

```

        {
            vert[0][0] = i;
            vert[0][1] = j;
            k++;
        }
    }
}

if(ways > 1)
{
    for(j = 0; j < k; j++)
    {
        i = vert[0][j];
        Graph[i][j] = 1;
        i = vert[1][j];
        Graph[i][j] = 1;
        i = 0;
    }
}
else if(ways == 1)
{
    for(j = 0; j < k; j++)
    {
        i = vert[0][j];
        Graph[i][j] = 1;
        i = vert[0][j + 1];
        Graph[i][j] = 1;
        i = 0;
    }
}

for(i = 0; i < size; i++)
{
    for(j = 0; j < k; j++)
        printf("%d", Graph[i][j]);
    printf("\n");
}

// Finding size of the graph
printf("Size of the graph: %d\n", graph_size);

// Finding isolated vertices
for(i = 0; i < size; i++)
{
    count = 0;
    for(j = 0; j < ways; j++)
        if(Graph[i][j] == 1) count++;

    if(count == 0) printf("Isolated vertex: %d\n", i+1);
}

// Finding end vertices
for(i = 0; i < size; i++)
{
    count = 0;
    for(j = 0; j < ways; j++)
    {
        if(Graph[i][j] == 1) count++;
    }
}

```

```
        if(count == 1) printf("End vertex: %d\n", i+1);
        else if(count == (size - 1)) printf("Dominant vertex: %d\n", i+1);
    }

    return 0;
}
```