Министерство науки и высшего образования

Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №5

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Определение характеристик графов"

Выполнили:

студенты группы 23ВВВ4

Королёв Д.В.

Алешин К.А.

Приняли:

Юрова О.В.

Деев М.В.

Пенза, 2024

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 $deg(v_i)$.

Задание 1.

Листинг

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>


int** create_adjacency_matrix(size_t size)
{
    int** matrix = (int**)malloc(size * sizeof(int*));
    if (!matrix)
    {
        return nullptr;
    }

    for (size_t i = 0; i < size; ++i)
    {
        matrix[i] = (int*)malloc(size * sizeof(int));
        if (!matrix[i])
        {
            for (size_t j = 0; j < i; ++j)
            {
                free(matrix[j]);

            }
            free(matrix);
            return nullptr;
        }
```

```
    }
    for (size_t i = 0; i < size; ++i)
    {
        for (size_t j = i + 1; j < size; ++j)
        {
            matrix[i][j] = rand() % 2;
        }
    }
    for (size_t i = 1; i < size; ++i)
    {
        for (size_t j = 0; j < i; ++j)
        {
            matrix[i][j] = matrix[j][i];
        }
    }

    for (size_t i = 0; i < size; ++i)
    {
        matrix[i][i] = 0;
    }

    return matrix;
}

void print_matrix(int** matrix, size_t size)
{
    for (size_t i = 0; i < size; ++i)
    {
        for (size_t j = 0; j < size; ++j) {

            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
}


void free_matrix(int** matrix, size_t size)
{
    if (!matrix)
```

```
    {
        return;
    }
    else
    {
        for (size_t i = 0; i < size; ++i)
        {
            free(matrix[i]);
        }
        free(matrix);
    }
}

size_t size_G_grapg(int** matrix, size_t size)
{
    size_t size_g = 0;
    for (size_t i = 0; i < size; ++i)
    {
        for (size_t j = i + 1; j < size; ++j)
        {
            if (matrix[i][j] == 1) ++size_g;
        }
    }
    return size_g;
}

void isolated(int** matrix, size_t size)
{
    size_t isol = 0;
    for (size_t i = 0; i < size; ++i)
    {
        int sum = 0;
        for (size_t j = 0; j < size; ++j)
        {
            sum += matrix[i][j] + matrix[j][i];
        }

        if (sum == 0) ++isol;
    }
    std::cout << "Isolated: " << isol << std::endl;
```

```
}

void terminal(int** matrix, size_t size)
{
    size_t term = 0;
    for (size_t i = 0; i < size; ++i)
    {
        int sum = 0;
        for (size_t j = 0; j < size; ++j)
        {
            sum += matrix[i][j] + matrix[j][i];
        }
        if (sum == 1) ++term;
    }
    std::cout << "Terminal: " << term << std::endl;
}

void dominating(int** matrix, size_t size)
{
    size_t dom = 0;
    for (size_t i = 0; i < size; ++i)
    {
        int sum = 0;
        for (size_t j = 0; j < size; ++j)
        {
            if (i != j)
            {
                sum += matrix[i][j];
            }
        }

        if (sum == size - 1)
        {
            ++dom;
        }
    }
    std::cout << "Dominating: " << dom << std::endl;
}
```

```
    void start()
    {

        srand(static_cast<unsigned int>(time(nullptr)));

        while (true)
        {
            std::cout << "if you wanna finish - enter
'0'\n";
            std::cout << "Enter size matrix (type size_t)
: ";
            size_t size;
            std::cin >> size;
            if (!size)
            {
                return;
            }
            int** matrix = create_adjacency_matrix(size);
            if (!matrix)
            {
                std::cerr << "Error allocate\n";
                continue;
            }
            print_matrix(matrix, size);
            std::cout << "Size G-praph - " <<
size_G_grapg(matrix, size) << std::endl;
            isolated(matrix, size);
            terminal(matrix, size);
            dominating(matrix, size);


            free_matrix(matrix, size);

            std::cout << std::endl;

            std::cin.clear();
        }
    }

    int main() {
```

```
    start();

}
```

Результат работы программы

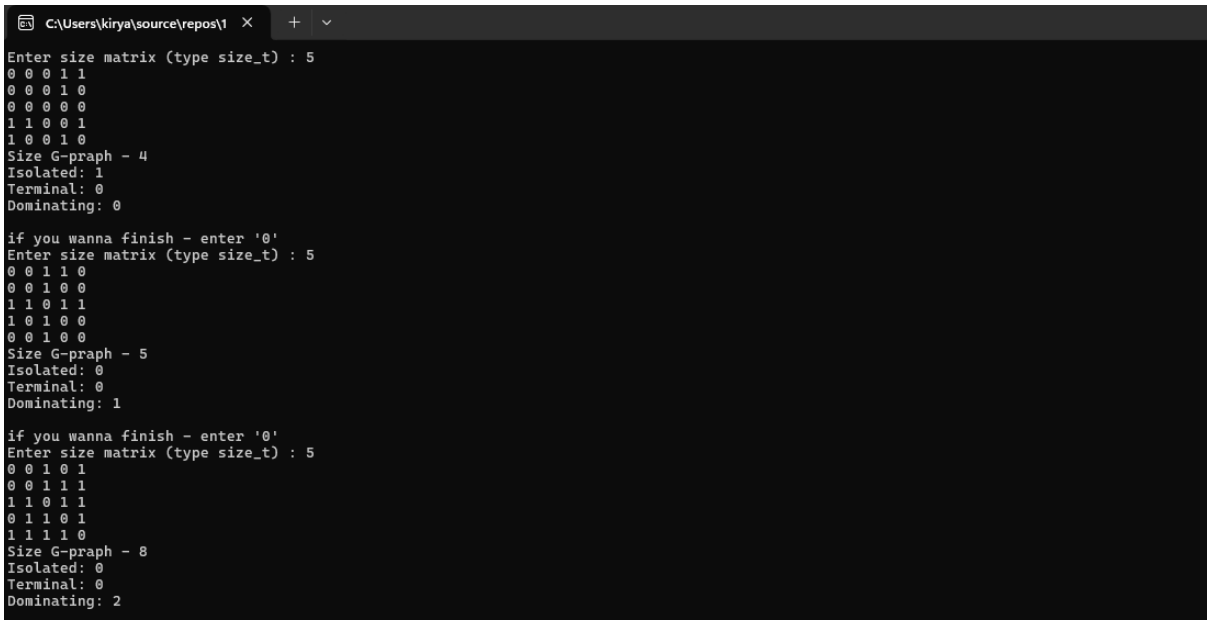

Рисунок № 1

Вывод - были получены навыки реализации неориентированных графов, Матрицы смежности, также получены навыки нахождения изолированных, доминирующих, концевых вершин.