МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯРОССИЙСКОЙ ФЕДЕРАЦИИПЕНЗЕНСКИЙ 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 24ВВВ3:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Изучение бинарного дерева.

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.

Пояснительный текст к программам

Эта программа выполняет комплексный анализ графов, работая как с неориентированными, так и с ориентированными графами.

Для неориентированного графа:

· Пользователь задаёт количество вершин, после чего генерируется симметричная матрица смежности

· Программа вычисляет и выводит размер графа (количество рёбер)

· Определяются степени вершин с учётом петель (каждая петля добавляет 2 к степени)

· Находится и выводится матрица инцидентности

· Выявляются специальные типы вершин: изолированные, концевые и доминирующие

Для ориентированного графа:

· Создаётся несимметричная матрица смежности, где направление рёбер задаётся от строки к столбцу

· Строится матрица инцидентности с обозначениями: 1 - начало дуги, -1 - конец дуги, 0 - отсутствие связи

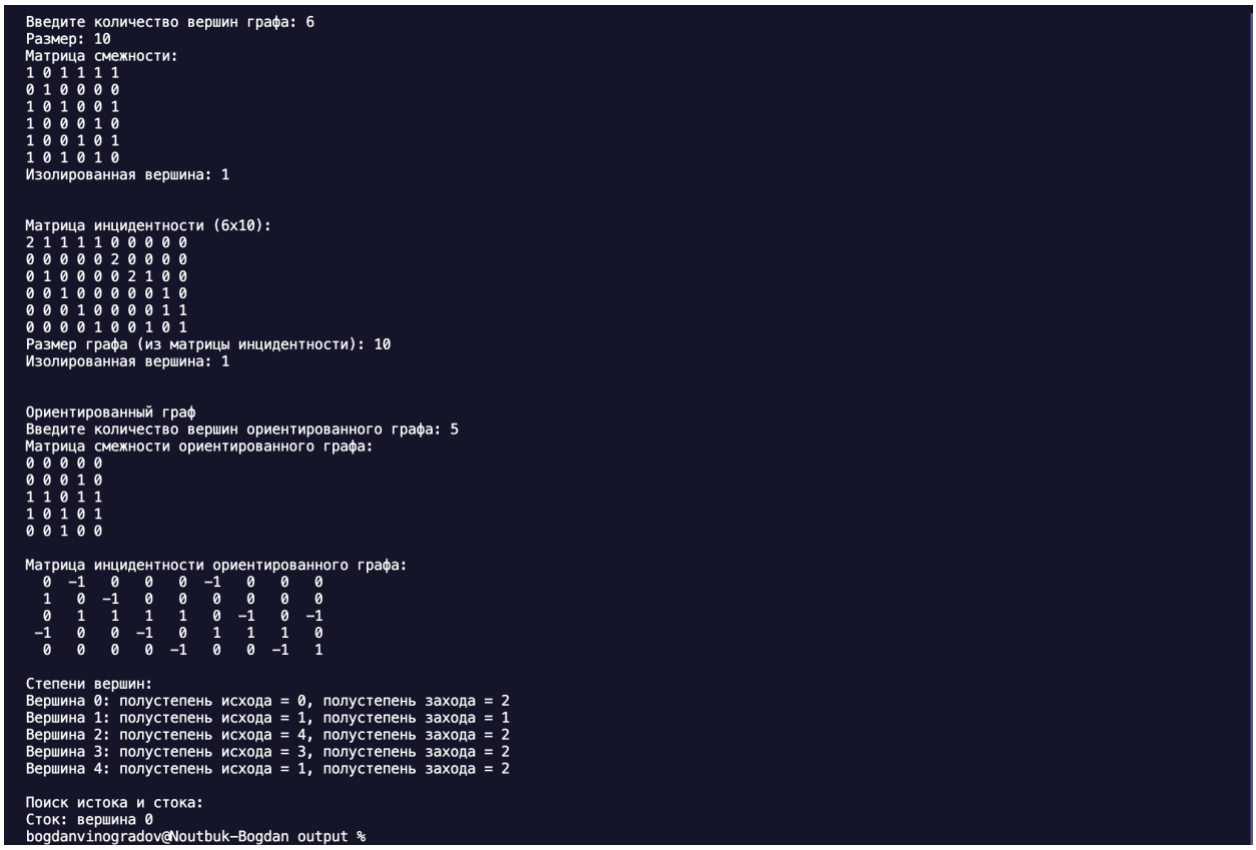
· Вычисляются полустепени исхода и захода для каждой вершины

· Определяются истоки (вершины с нулевой полустепенью захода) и стоки (вершины с нулевой полустепенью исхода)

Программа демонстрирует различные способы представления графов (матрицы смежности и инцидентности) и показывает, как выполнять анализ свойств вершин на основе этих представлений. Все структуры данных корректно освобождаются после использования.

Результаты программ

1 Рис. - Результат работы **lab5.cpp**



Вывод: в ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №5 – определение характеристик графов.

Приложение А

Листинг

Файл lab5.cpp

```
/*выводить степень вершин, нахождение элементов в
ориентированном графе*/
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <locale>
#include <limits>

void clearScreen();
int isInteger(const std::string& message);

int main() {
    setlocale(LC_ALL, "Rus");
    clearScreen();
    srand(time(NULL));
    int** G;
    int N, size = 0;
    int* loop, *deg;

    N = isInteger("Введите количество вершин графа: ");
    while (N <= 0) {
        std::cout << "Ошибка! Количество вершин должно быть
положительным\n";
        N = isInteger("Введите количество вершин графа: ");
    }

    G = (int**)malloc(sizeof(int*) * N);
    loop = (int*)malloc(N * sizeof(int));
    deg = (int*)malloc(N * sizeof(int));

    for (int i = 0; i < N; i++) {
        G[i] = (int*)malloc(N * sizeof(int));
        loop[i] = deg[i] = 0;
    }

    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            G[i][j] = G[j][i] = rand() % 2;
            size += G[i][j];
        }
    }

    std::cout << "Размер: " << size << "\n";

    std::cout << "Матрица смежности:\n";
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
            std::cout << G[i][j] << " ";
        }
        std::cout << "\n";
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i == j) {
                loop[i] = G[i][j];
            }
            else {
                deg[i] += G[i][j];
            }
        }
        deg[i] += loop[i] * 2;
    }

    for (int i = 0; i < N; i++){
        if ((loop[i] == 0 && deg[i] == 0) || (loop[i] == 1 &&
deg[i] == 2)) {
            std::cout << "Изолированная вершина: " << i << "\n";
        }
        if ((deg[i] - 2 * loop[i]) == (N - 1)) {
            std::cout << "Доминирующая вершина: " << i << "\n";
        }
        if (deg[i] == 1) {
            std::cout << "Концевая вершина: " << i << "\n";
        }
    }

    int edgeCount = size;

    int** incidence = (int**)malloc(sizeof(int*) * N);
    for (int i = 0; i < N; i++) {
        incidence[i] = (int*)malloc(edgeCount * sizeof(int));
        for (int j = 0; j < edgeCount; j++) {
            incidence[i][j] = 0;
        }
    }

    int currentEdge = 0;
    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            if (G[i][j] == 1) {
                if (i == j) {
                    incidence[i][currentEdge] = 2;
                } else {
                    incidence[i][currentEdge] = 1;
                    incidence[j][currentEdge] = 1;
                }
                currentEdge++;
            }
        }
```

```
    }

    std::cout << "\n\nМатрица инцидентности (" << N << "x" <<
edgeCount << "):\n";
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < edgeCount; j++) {
            std::cout << incidence[i][j] << " ";
        }
        std::cout << "\n";
    }


    int sizeFromIncidence = edgeCount;
    std::cout << "Размер графа (из матрицы инцидентности): " <<
sizeFromIncidence << "\n";


    int* degInc = (int*)malloc(N * sizeof(int));
    int* loopInc = (int*)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++) {
        degInc[i] = 0;
        loopInc[i] = 0;
        for (int j = 0; j < edgeCount; j++) {
            degInc[i] += incidence[i][j];
            if (incidence[i][j] == 2) {
                loopInc[i] = 1;
            }
        }
    }

    for (int i = 0; i < N; i++) {
        if ((loopInc[i] == 0 && degInc[i] == 0) || (loopInc[i]
== 1 && degInc[i] == 2)) {
            std::cout << "Изолированная вершина: " << i << "\n";
        }
    }

    for (int i = 0; i < N; i++) {
        if (degInc[i] == 1 && loopInc[i] == 0) {
            std::cout << "Концевая вершина: " << i << "\n";
        }
    }

    for (int i = 0; i < N; i++) {
        int connections = 0;
        for (int j = 0; j < N; j++) {
            if (i != j) {
                bool connected = false;
                for (int k = 0; k < edgeCount; k++) {
                    if ((incidence[i][k] == 1 && incidence[j][k]
== 1) ||
                        (i == j && incidence[i][k] == 2)) {
                        connected = true;
```

```
                        break;
                    }
                }
                if (connected) connections++;
            }
        }
        if (connections == N - 1) {
            std::cout << "Доминирующая вершина: " << i << "\n";
        }
    }

    free(loopInc);
    free(degInc);
    for (int i = 0; i < N; i++) {
        free(incidence[i]);
    }
    free(incidence);
    free(loop);
    free(deg);
    for (int i = 0; i < N; i++) {
        free(G[i]);
    }
    free(G);

    std::cout << "\n\nОриентированный граф\n";

    int orientN;
    orientN = isInteger("Введите количество вершин
ориентированного графа: ");
    while (orientN <= 0) {
        std::cout << "Ошибка! Количество вершин должно быть
положительным\n";
        orientN = isInteger("Введите количество вершин
ориентированного графа: ");
    }

    int** orientG = (int**)malloc(sizeof(int*) * orientN);
    for (int i = 0; i < orientN; i++) {
        orientG[i] = (int*)malloc(orientN * sizeof(int));
        for (int j = 0; j < orientN; j++) {
            orientG[i][j] = 0;
        }
    }

    int orientEdgeCount = 0;
    for (int i = 0; i < orientN; i++) {
        for (int j = 0; j < orientN; j++) {
            if (i != j) {
                orientG[i][j] = rand() % 2;
                if (orientG[i][j] == 1) orientEdgeCount++;
            }
        }
    }
```

```
    std::cout << "Матрица смежности ориентированного графа:\n";
    for (int i = 0; i < orientN; i++) {
        for (int j = 0; j < orientN; j++) {
            std::cout << orientG[i][j] << " ";
        }
        std::cout << "\n";
    }

    int** orientIncidence = (int**)malloc(sizeof(int*) *
orientN);
    for (int i = 0; i < orientN; i++) {
        orientIncidence[i] = (int*)malloc(orientEdgeCount *
sizeof(int));
        for (int j = 0; j < orientEdgeCount; j++) {
            orientIncidence[i][j] = 0;
        }
    }

    int orientCurrentEdge = 0;
    for (int i = 0; i < orientN; i++) {
        for (int j = 0; j < orientN; j++) {
            if (orientG[i][j] == 1 && i != j) {
                orientIncidence[i][orientCurrentEdge] = 1;
                orientIncidence[j][orientCurrentEdge] = -1;
                orientCurrentEdge++;
            }
        }
    }

    std::cout << "\nМатрица инцидентности ориентированного
графа:\n";
    for (int i = 0; i < orientN; i++) {
        for (int j = 0; j < orientEdgeCount; j++) {
            if (orientIncidence[i][j] == 1) {
                std::cout << "  1 ";
            } else if (orientIncidence[i][j] == -1) {
                std::cout << " -1 ";
            } else {
                std::cout << "  0 ";
            }
        }
        std::cout << "\n";
    }

    int* inDeg = (int*)malloc(orientN * sizeof(int));
    int* outDeg = (int*)malloc(orientN * sizeof(int));

    for (int i = 0; i < orientN; i++) {
        inDeg[i] = 0;
        outDeg[i] = 0;
        for (int j = 0; j < orientEdgeCount; j++) {
            if (orientIncidence[i][j] == 1) outDeg[i]++;
```

```
            if (orientIncidence[i][j] == -1) inDeg[i]++;
        }
    }

    std::cout << "\nСтепени вершин:\n";
    for (int i = 0; i < orientN; i++) {
        std::cout << "Вершина " << i << ": полустепень исхода =
" << outDeg[i]
                  << ", полустепень захода = " << inDeg[i] <<
"\n";
    }

    std::cout << "\nПоиск истока и стока:\n";
    for (int i = 0; i < orientN; i++) {
        if (inDeg[i] == 0 && outDeg[i] > 0) {
            std::cout << "Исток: вершина " << i << "\n";
        }
        if (outDeg[i] == 0 && inDeg[i] > 0) {
            std::cout << "Сток: вершина " << i << "\n";
        }
        if (inDeg[i] == 0 && outDeg[i] == 0) {
            std::cout << "Изолированная вершина: " << i << "\n";
        }
    }

    free(inDeg);
    free(outDeg);
    for (int i = 0; i < orientN; i++) {
        free(orientIncidence[i]);
        free(orientG[i]);
    }
    free(orientIncidence);
    free(orientG);

    return 0;
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int isInteger(const std::string& message) {
    int value;
    while (true) {
        std::cout << message;
        if (!(std::cin >> value)) {
            std::cout << "Ошибка: введено не число.\n";
            std::cin.clear();
```

```
std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            continue;
        }
        if (std::cin.peek() != '\n') {
            std::cout << "Ошибка: введено не целое число.\n";
            std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            continue;
        }
        return value;
    }
     }
```