Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

**Отчёт**

по лабораторной работе №10

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “ Поиск расстояний в взвешенном графе”

Выполнили:
Студенты группы 24ВВВ4
Чернышевский Е.И.
Суходолов И.А.

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

Пенза 2025

**Название:**
Поиск расстояний в взвешенном графе

**Цель работы:**
Разработать программу, где реализуется поиск того или иного значения с клавиатуры

**Лабораторное задание:**

**Задание 1**

**1.** Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.

**2.** Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки C++.

**3.** Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

**Задание 2**

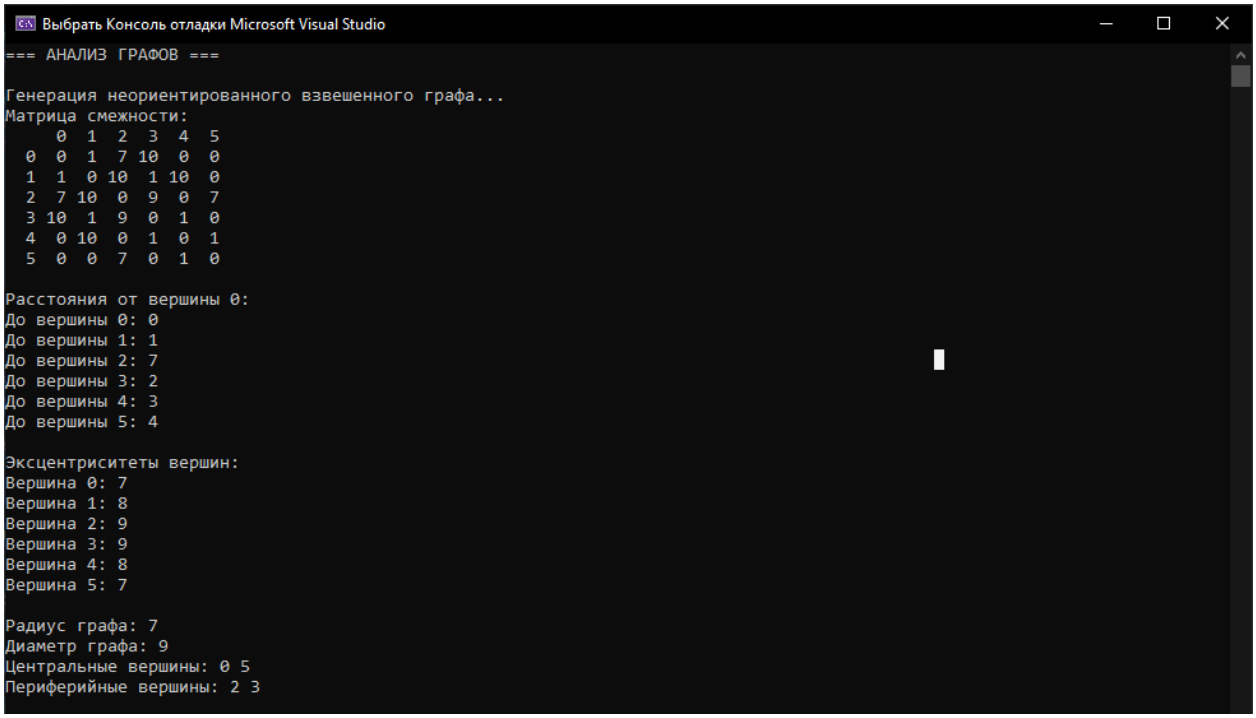
**1.** Для каждого из вариантов сгенерированных графов (ориентированного и не ориентированного) определите радиус и диаметр.

**2.** Определите подмножества периферийных и центральных вершин.

**Задание 3**

**1.** Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже). В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

**Результат работы программы:**




**Выводы:**

В ходе выполнения лабораторной работы были разработаны программы, выполняющие указанные в лабораторной работе задачи. Результаты работ программ совпали с результатами трассировок, следовательно, программы работают без ошибок.

Получили опыт в создании проектов в среде MicrosoftVisualStudio.

# Листинг:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <queue>
#include <vector>
#include <algorithm>
#include <limits.h>
#include <locale.h>

using namespace std;

// Алгоритм Дейкстры для взвешенного графа
void dijkstra(int** G, int numG, int* dist, int s) {
    vector<bool> visited(numG, false);

    for (int i = 0; i < numG; i++) {
        dist[i] = INT_MAX;
    }
    dist[s] = 0;

    for (int count = 0; count < numG - 1; count++) {
        // Находим вершину с минимальным расстоянием
        int minDist = INT_MAX;
        int minIndex = -1;

        for (int v = 0; v < numG; v++) {
            if (!visited[v] && dist[v] <= minDist) {
                minDist = dist[v];
                minIndex = v;
            }
        }

        if (minIndex == -1) break;

        visited[minIndex] = true;

        // Обновляем расстояния до смежных вершин
        for (int v = 0; v < numG; v++) {
            if (!visited[v] && G[minIndex][v] != 0 &&
                dist[minIndex] != INT_MAX &&
                dist[minIndex] + G[minIndex][v] < dist[v]) {
                dist[v] = dist[minIndex] + G[minIndex][v];
            }
        }
    }
}

// BFS для ненагруженного графа (оставляем для сравнения)
void BFSD(int** G, int numG, int* dist, int s) {
    queue<int> q;
    int v;

    for (int i = 0; i < numG; i++) {
        dist[i] = -1;
    }

    dist[s] = 0;
    q.push(s);

    while (!q.empty()) {
```

```
            v = q.front();
            q.pop();

            for (int i = 0; i < numG; i++) {
                if (G[v][i] != 0 && dist[i] == -1) {
                    q.push(i);
                    dist[i] = dist[v] + 1;
                }
            }
        }
    }

    void printMatrix(int** G, int numG) {
        printf("Матрица смежности:\n");
        for (int i = 0; i < numG; i++) {
            for (int j = 0; j < numG; j++) {
                printf("%3d", G[i][j]);
            }
            printf("\n");
        }
    }

    void findGraphProperties(int** G, int numG, bool directed) {
        int** distanceMatrix = (int**)malloc(numG * sizeof(int*));
        for (int i = 0; i < numG; i++) {
            distanceMatrix[i] = (int*)malloc(numG * sizeof(int));
        }

        // Вычисляем матрицу расстояний с учетом весов (алгоритм Дейкстры)
        printf("Вычисление матрицы расстояний...\n");
        for (int i = 0; i < numG; i++) {
            dijkstra(G, numG, distanceMatrix[i], i);
        }

        printf("\nМатрица расстояний (с учетом весов):\n");
        printf("    ");
        for (int i = 0; i < numG; i++) {
            printf("%3d", i);
        }
        printf("\n");

        for (int i = 0; i < numG; i++) {
            printf("%2d:", i);
            for (int j = 0; j < numG; j++) {
                if (distanceMatrix[i][j] == INT_MAX) {
                    printf("  ∞");
                }
                else {
                    printf("%3d", distanceMatrix[i][j]);
                }
            }
            printf("\n");
        }

        // Находим эксцентриситеты вершин
        int* eccentricity = (int*)malloc(numG * sizeof(int));
        int radius = INT_MAX;
        int diameter = 0;

        for (int i = 0; i < numG; i++) {
            eccentricity[i] = 0;
            for (int j = 0; j < numG; j++) {
                if (i != j && distanceMatrix[i][j] != INT_MAX &&
                    distanceMatrix[i][j] > eccentricity[i]) {
                    eccentricity[i] = distanceMatrix[i][j];
```

```
        }
    }
    // Если вершина изолирована (все расстояния = ∞), эксцентриситет = 0
    if (eccentricity[i] == 0) {
        eccentricity[i] = INT_MAX;
    }
}

// Находим радиус и диаметр (игнорируя несвязные вершины)
for (int i = 0; i < numG; i++) {
    if (eccentricity[i] != INT_MAX) {
        if (eccentricity[i] < radius) {
            radius = eccentricity[i];
        }
        if (eccentricity[i] > diameter) {
            diameter = eccentricity[i];
        }
    }
}

// Если граф несвязный
if (radius == INT_MAX) {
    radius = -1;
    diameter = -1;
}


for (int i = 0; i < numG; i++) {
    if (eccentricity[i] == INT_MAX) {
        printf("\n");
    }
    else {
        printf("\n");
    }
}

if (radius != -1) {
    printf("\nРадиус графа: %d\n", radius);
    printf("Диаметр графа: %d\n", diameter);

    // Находим центральные вершины

    bool hasCenter = false;
    for (int i = 0; i < numG; i++) {
        if (eccentricity[i] != INT_MAX && eccentricity[i] == radius) {

            hasCenter = true;
        }
    }
    if (!hasCenter) printf("нет");


    // Находим периферийные вершины

    bool hasPeripheral = false;
    for (int i = 0; i < numG; i++) {
        if (eccentricity[i] != INT_MAX && eccentricity[i] == diameter) {

            hasPeripheral = true;
        }
    }
    if (!hasPeripheral) printf("нет");

}
else {
```

```
        printf("\nГраф несвязный! Невозможно определить радиус и диаметр.\n");
    }

    // Освобождаем память
    for (int i = 0; i < numG; i++) {
        std::free(distanceMatrix[i]);
    }
    std::free(distanceMatrix);
    std::free(eccentricity);
}

int main() {
    setlocale(LC_ALL, "rus");
    int** G;
    int numG, current;
    int* dist;

    srand(time(NULL));

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

    dist = (int*)malloc(numG * sizeof(int));
    G = (int**)malloc(numG * sizeof(int*));
    for (int i = 0; i < numG; i++) {
        G[i] = (int*)malloc(numG * sizeof(int));
    }

    // Задание 1: Генерация неориентированного взвешенного графа
    printf("\n=== ЗАДАНИЕ 1 ===\n");

    // Генерируем неориентированный взвешенный граф
    printf("Генерация неориентированного взвешенного графа...\n");
    for (int i = 0; i < numG; i++) {
        for (int j = i; j < numG; j++) {
            if (i == j) {
                G[i][j] = 0;
            }
            else {
                // Случайный вес от 0 до 10 (0 означает отсутствие ребра)
                // Увеличиваем вероятность ненулевых весов для лучшей связности
                int weight = (rand() % 10 == 0) ? 0 : (rand() % 10 + 1);
                G[i][j] = G[j][i] = weight;
            }
        }
    }

    printMatrix(G, numG);

    printf("\nВведите стартовую вершину для поиска расстояний: ");
    scanf("%d", &current);

    printf("\nРасстояния от вершины %d (алгоритм Дейкстры):\n", current);
    dijkstra(G, numG, dist, current);
    for (int i = 0; i < numG; i++) {
        if (dist[i] == INT_MAX) {
            printf("До вершины %d: недостижима\n", i);
        }
        else {
            printf("До вершины %d: %d\n", i, dist[i]);
        }
    }

    // Задание 2: Анализ графа
    printf("\n=== ЗАДАНИЕ 2 ===\n");
```

```
    // Анализ неориентированного графа
    printf("\nАнализ неориентированного взвешенного графа:\n");
    findGraphProperties(G, numG, false);

    // Генерируем ориентированный граф для сравнения
    printf("\n--- Ориентированный взвешенный граф ---\n");
    int** directedG = (int**)malloc(numG * sizeof(int*));
    for (int i = 0; i < numG; i++) {
        directedG[i] = (int*)malloc(numG * sizeof(int));
        for (int j = 0; j < numG; j++) {
            if (i == j) {
                directedG[i][j] = 0;
            }
            else {
                // Для ориентированного графа ребра могут быть несимметричными
                int weight = (rand() % 10 == 0) ? 0 : (rand() % 10 + 1);
                directedG[i][j] = weight;
            }
        }
    }

    printf("\nМатрица смежности ориентированного графа:\n");
    printMatrix(directedG, numG);

    printf("\nАнализ ориентированного взвешенного графа:\n");
    findGraphProperties(directedG, numG, true);

    // Освобождаем память
    for (int i = 0; i < numG; i++) {
        std::free(G[i]);
        std::free(directedG[i]);
    }
    std::free(G);
    std::free(directedG);
    std::free(dist);

    printf("\nНажмите любую клавишу для выхода...");
    _getch();

    return 0;
}
```