МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

ПРЕДСТАВЛЕНИЕ ГРАФИКОВ. МАТРИЦА ИНЦИДЕНТНОСТИ.

Отчёт по лабораторной работе №3

по дисциплине

«Дискретная математика»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Проверила преподаватель _____/Пахарева И. В./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цель

Цель работы: изучение основ теории графов, представление графов в виде матрицы инцидентности.

Задание

– По матрице инциндентности определить кол-во двунаправленных дуг графа (матрицу инцидентности задать из файла).

– Вывести множество найденных дуг (согласно номерам вершин).

Решение

Схема алгоритма решения представлена на рисунке 1. Исходный код решений представлен в Приложении А1.

Рисунок 1 - Схема алгоритма Задания.

Вывод

В ходе выполнения лабораторной работы была реализована программа, которая читает матрицу инцидентности из файла «input.txt» и определяет кол-во двунаправленных дуг графа.

Приложение А1. Исходный код

```rust
use std::collections::{HashMap, HashSet};
use std::env;
use std::fs::File;
use std::io::{self, BufRead, BufReader};

// Модуль для работы с Graphviz (визуализация графов)
mod graphviz;

fn main() {
    // Проверяем, включена ли визуализация графа через аргументы командной строки
    let enable_visualization = {
        let args: Vec<String> = env::args().collect();
        args.contains(&"-viz".to_string()) || args.contains(&"--visualize".to_string())
    };

    // Читаем матрицу смежности из файла "input.txt"
    let (edges, _edge_labels) = read_matrix("input.txt").expect("Ошибка чтения файла");

    // Если визуализация включена, сохраняем граф в формате DOT
    if enable_visualization {
        if let Err(e) = graphviz::save_dot(&edges, "graph.dot") {
            eprintln!("Ошибка сохранения графа: {}", e);
            return;
        }
    } else {
        println!("Визуализация отключена");
    }

    // Поиск двунаправленных дуг
    let _edge_set: HashSet<(usize, usize)> = edges.iter().map(|(u, v, _)| (*u, *v)).collect
    let mut result_pairs = HashSet::new();
```

```
    // Создаём обратные рёбра для быстрого поиска
    let reverse_edges: HashMap<(usize, usize), String> = edges
        .iter()
        .map(|(u, v, label)| ((*v, *u), label.clone()))
        .collect();

    // Проверяем каждое ребро на наличие обратного
    for (u, v, label) in &edges {
        if let Some(reverse_label) = reverse_edges.get(&(*u, *v)) {
            // Исключаем петли (u == v) и рёбра с одинаковыми метками
            if *u != *v && label != reverse_label {
                // Упорядочиваем вершины, чтобы избежать дубликатов (A-B и B-A)
                let pair = if *u < *v { (*u, *v) } else { (*v, *u) };
                result_pairs.insert((pair, label.clone(), reverse_label.clone()));
            }
        }
    }

    // Удаляем дубликаты (A-B и B-A считаем одной парой)
    let mut unique_pairs = HashMap::new();
    for (pair, label1, label2) in result_pairs {
        unique_pairs.entry(pair).or_insert((label1, label2));
    }

    // Выводим результаты
    println!("\nКоличество двунаправленных дуг: {}", unique_pairs.len());
    println!("Множество найденных дуг:");
    for ((u, v), (label1, label2)) in unique_pairs {
        println!("({}, {}) - рёбра {} и {}", u, v, label1, label2);
    }
}

// Функция для чтения матрицы смежности из файла
fn read_matrix(
```

```
    filename: &str,
) -> Result<(Vec<(usize, usize, String)>, HashMap<String, (usize, usize)>), io::Error> {
    let file = File::open(filename)?; // Открываем файл
    let reader = BufReader::new(file); // Создаём буферизированный ридер
    let mut lines = reader.lines();

    // Читаем заголовок файла (метки рёбер)
    let header = lines
        .next()
        .ok_or(io::Error::new(io::ErrorKind::InvalidData, "Нет заголовка"))??;
    let edge_labels: Vec<String> = header
        .split_whitespace()
        .map(|s| s.trim().to_string())
        .filter(|s| !s.is_empty())
        .collect();

    let mut edges = Vec::new(); // Список рёбер
    let mut edge_map = HashMap::new(); // Карта меток рёбер
    let mut vertex_map = HashMap::new(); // Карта вершин
    let mut vertex_count = 0; // Счётчик вершин

    // Обрабатываем строки файла
    for line in lines {
        let line = line?;
        let parts: Vec<&str> = line.split_whitespace().collect();
        if parts.is_empty() {
            continue; // Пропускаем пустые строки
        }

        // Проверяем, совпадает ли количество столбцов с количеством меток рёбер
        if parts.len() - 1 != edge_labels.len() {
            return Err(io::Error::new(
                io::ErrorKind::InvalidData,
                format!(
```

```
                "Неверное количество столбцов. Ожидалось {}, получено {}",
                edge_labels.len(),
                parts.len() - 1
            ),
        ));
    }

    let vertex_name = parts[0]; // Имя вершины
    if !vertex_map.contains_key(vertex_name) {
        vertex_count += 1; // Присваиваем новый номер вершине
        vertex_map.insert(vertex_name.to_string(), vertex_count);
    }
    let vertex_num = vertex_map[vertex_name]; // Получаем номер вершины

    // Обрабатываем значения в строке
    for (j, &value) in parts.iter().skip(1).enumerate() {
        if j >= edge_labels.len() {
            return Err(io::Error::new(
                io::ErrorKind::InvalidData,
                format!("Индекс ребра {} выходит за пределы {}", j, edge_labels.len())
            ));
        }

        let edge_label = &edge_labels[j];
        let num: i32 = value
            .parse()
            .map_err(|e| io::Error::new(io::ErrorKind::InvalidData, e))?;

        match num {
            2 => {
                // Петля (ребро начинается и заканчивается в одной вершине)
                edge_map.insert(edge_label.clone(), (vertex_num, vertex_num));
            }
            1 => {
```

```
                    // Начало дуги
                    let current_end = edge_map.get(edge_label).map_or(0, |v| v.1);
                    edge_map.insert(edge_label.clone(), (vertex_num, current_end));
                }
                -1 => {
                    // Конец дуги
                    let current_start = edge_map.get(edge_label).map_or(0, |v| v.0);
                    edge_map.insert(edge_label.clone(), (current_start, vertex_num));
                }
                _ => {}
            }
        }
    }

    // Преобразуем карту рёбер в список
    for (label, (u, v)) in &edge_map {
        edges.push((*u, *v, label.clone()));
    }

    Ok((edges, edge_map)) // Возвращаем список рёбер и карту рёбер
}
```