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 представления ориентированных графов в виде матрицы инцидентности и алгоритмический анализ этой структуры для поиска двунаправленных дуг.

Задание

– По матрице инцидентности, заданной в файле `input.txt`, определить количество двунаправленных дуг графа.

– Вывести множество найденных дуг (по номерам вершин), то есть пары вершин, между которыми имеются ребра в обоих направлениях.

Решение

Схема алгоритма решения представлена на рисунке 1. Пример работы программы представлен на рисунках 2.1 и 2.2. Исходный код решений представлен в Приложениях А1, А2 и А3.

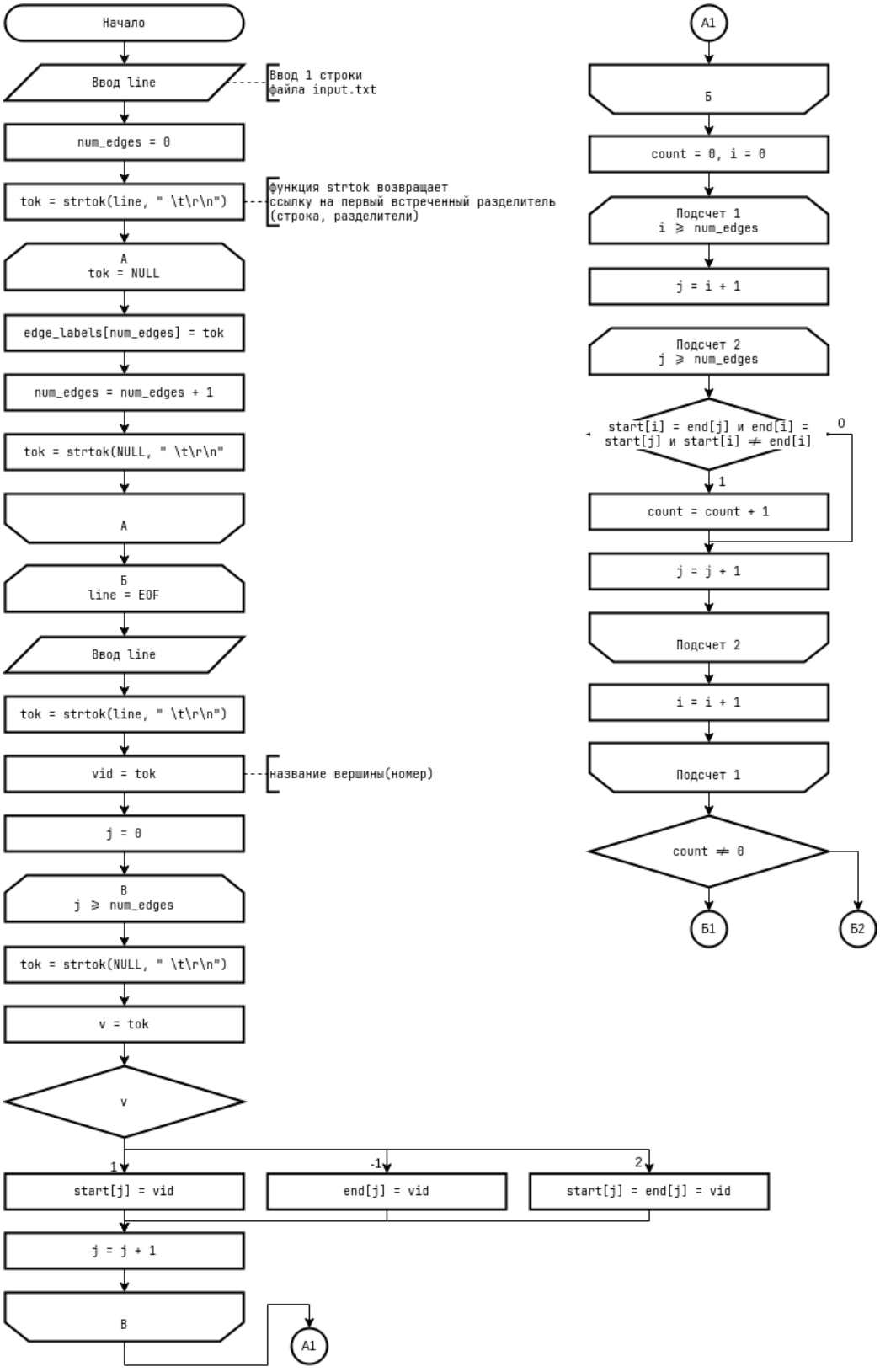

Рисунок 1.1 - Схема алгоритма основной программы стр.1.

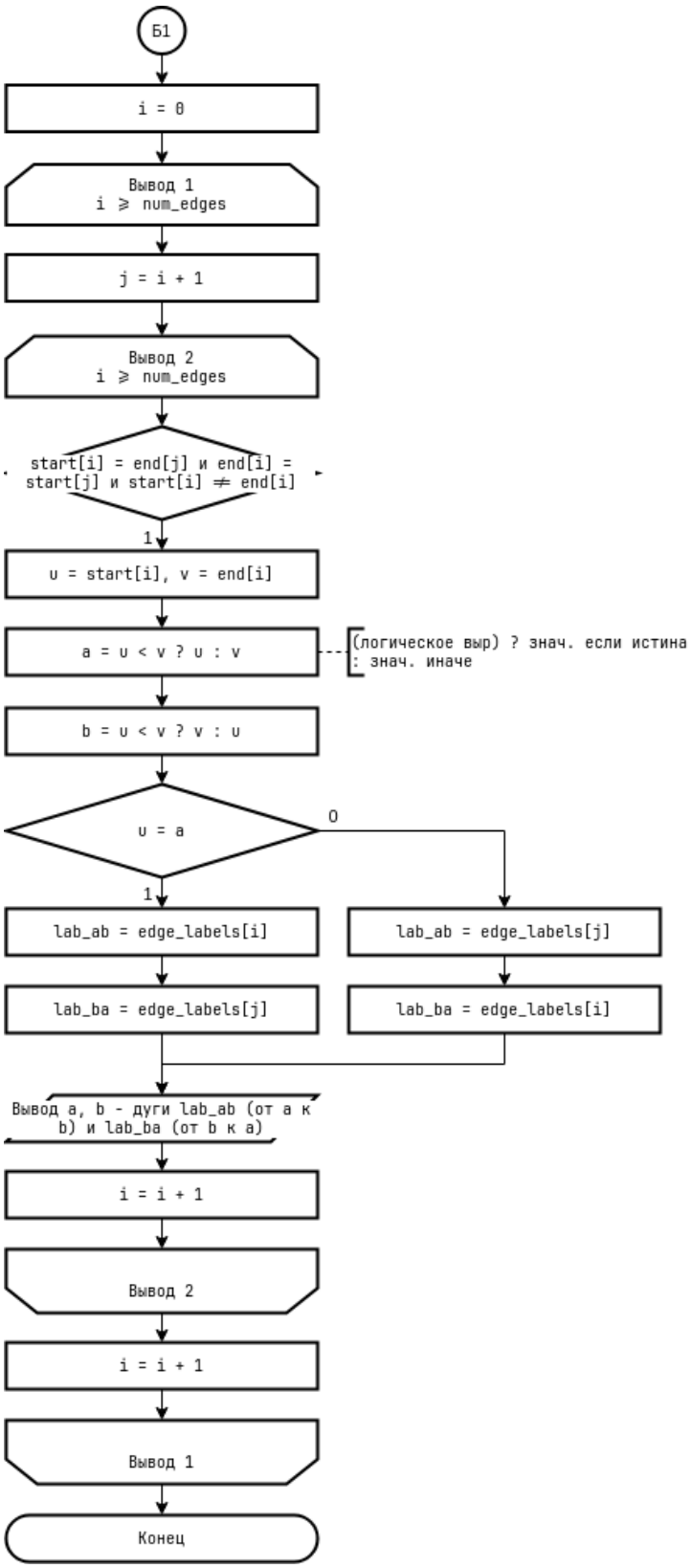

Рисунок 1.2 - Схема алгоритма основной программы стр.2.

```
❯ cargo run --release -- -viz
    Finished `release` profile [optimized] target(s) in 0.00s
     Running `target/release/lab3 -viz`
Граф сохранён в файл: graph.dot

Количество двунаправленных дуг: 1
Множество найденных дуг:
(3, 4) – рёбра Q и P
❯ dot -Tpng graph.dot -o graph.png
```

Рисунок 2.1 - Пример работы программы.

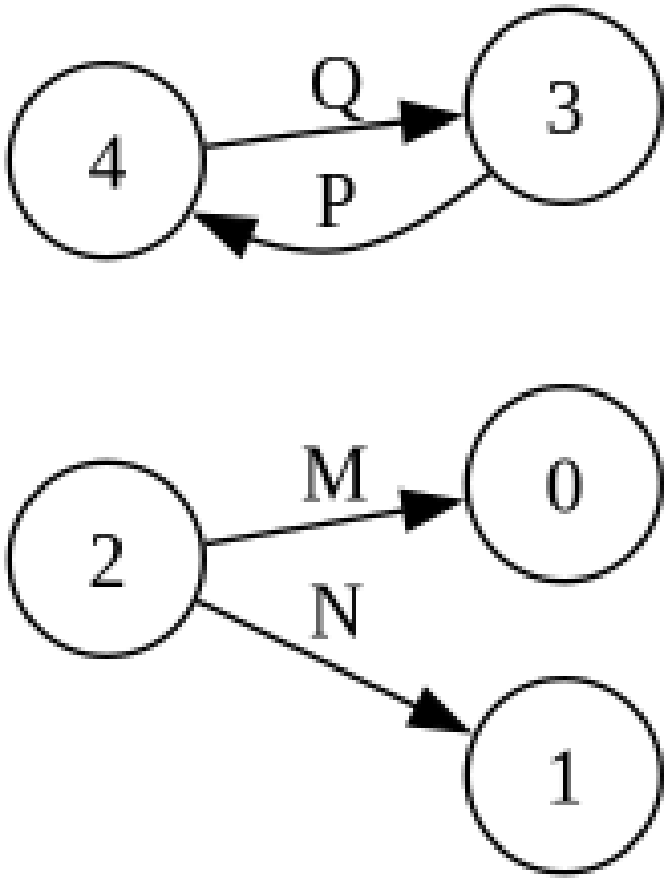

Рисунок 2.2 - Сгенерированный по входному файлу input.txt граф.

Вывод

В ходе выполнения данной лабораторной работы была разработана и отлажена программа на языке Rust, которая:

- Считывает матрицу инцидентности из текстового файла `input.txt`.
- Корректно разбирает каждый столбец-метку для выявления начальной и конечной вершины дуги.
- Определяет двунаправленные дуги (пары вершин, между которыми существуют дуги в обоих направлениях), исключая петли и одинаковые метки.
- Выводит количество таких двунаправленных дуг и подробную информацию о каждом ребре (номера вершин и соответствующие метки).

Приложение А1. Исходный код

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

mod graphviz; // Модуль для визуализации графа

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

    // Читаем матрицу инцидентности из файла "input.txt"
    let (edges, _edge_labels) = read_incidence_matrix("input.txt").expect("Ошибка чтения фа

    // 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. Для включения используйте флаг --visualize или -v
    }

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

```rust
let mut found_bidirectional_arcs = HashSet::new();

// Проверяем каждое ребро на наличие обратного
for (u, v, label) in &edges {
    // Ищем ребро из v в u
    if let Some(reverse_label) = reverse_edges.get(&(*v, *u)) {
        // Исключаем петли (u == v) и убеждаемся, что это разные дуги
        if *u != *v {
            // Упорядочиваем вершины, чтобы избежать дубликатов (A-B и B-A)
            let ordered_pair = if *u < *v { (*u, *v) } else { (*v, *u) };
            // Добавляем найденную двунаправленную дугу
            found_bidirectional_arcs.insert((ordered_pair, label.clone(), reverse_label
        }
    }
}

// Удаляем дубликаты для вывода (A-B и B-A считаем одной парой)
let mut unique_bidirectional_arcs_for_output = HashMap::new();
for ((u, v), label1, label2) in found_bidirectional_arcs {
    unique_bidirectional_arcs_for_output.entry((u, v)).or_insert((label1, label2));
}

// Выводим результаты
println!("\n--- Результаты ---");
println!("Количество двунаправленных дуг: {}", unique_bidirectional_arcs_for_output.len
println!("Множество найденных дуг:");
if unique_bidirectional_arcs_for_output.is_empty() {
    println!("  Двунаправленные дуги не найдены.");
} else {
    for ((u, v), (label_uv, label_vu)) in unique_bidirectional_arcs_for_output {
        println!("  ({}, {}) - дуги \"{}\" (от {} к {}) и \"{}\" (от {} к {})", u, v,
    }
}
```

```
    println!("------------------\n");
}

/// Функция для чтения матрицы инцидентности из файла.
/// Возвращает список дуг (начало, конец, метка) и карту меток дуг.
fn read_incidence_matrix(
    filename: &str,
) -> Result<(Vec<(usize, usize, String)>, HashMap<String, (usize, usize)>), io::Error> {
    let file = File::open(filename)?;
    let reader = BufReader::new(file);
    let mut lines = reader.lines();

    // Читаем метки дуг из заголовка файла
    let header = lines
        .next()
        .ok_or(io::Error::new(io::ErrorKind::InvalidData, "Файл пуст или отсутствует заголо
    let edge_labels: Vec<String> = header
        .split_whitespace()
        .map(|s| s.trim().to_string())
        .filter(|s| !s.is_empty())
        .collect();

    let mut edges_list = Vec::new(); // Список дуг
    let mut temp_edge_data: HashMap<String, (usize, usize)> = HashMap::new(); // Временна
    let mut vertex_name_to_id = HashMap::new(); // Карта имен вершин в ID
    let mut next_vertex_id = 1; // Счётчик ID вершин

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

```
let vertex_name = parts[0]; // Имя вершины
// Получаем или генерируем числовой ID для вершины
let current_vertex_id = *vertex_name_to_id.entry(vertex_name.to_string()).or_insert
    let id = next_vertex_id;
    next_vertex_id += 1;
    id
});

// Проверяем количество столбцов
if parts.len() - 1 != edge_labels.len() {
    return Err(io::Error::new(
        io::ErrorKind::InvalidData,
        format!(
            "Неверное количество столбцов для вершины '{}'. Ожидалось {}, получено
            vertex_name,
            edge_labels.len(),
            parts.len() - 1
        ),
    ));
}

// Обрабатываем значения для каждой дуги в строке
for (j, &value_str) in parts.iter().skip(1).enumerate() {
    let edge_label = &edge_labels[j];
    let incidence_value: i32 = value_str
        .parse()
        .map_err(|e| io::Error::new(io::ErrorKind::InvalidData, format!("Неверное з

    let (mut start_node, mut end_node) = temp_edge_data.get(edge_label).cloned().un

    match incidence_value {
        1 => { // Начало дуги
            start_node = current_vertex_id;
```

```
                }
                -1 => { // Конец дуги
                    end_node = current_vertex_id;
                }
                2 => { // Петля
                    start_node = current_vertex_id;
                    end_node = current_vertex_id;
                }
                0 => { /* Нет связи */ }
                _ => { // Недопустимое значение
                    return Err(io::Error::new(
                        io::ErrorKind::InvalidData,
                        format!(
                            "Недопустимое значение '{}' в матрице инцидентности для дуги '
                            incidence_value, edge_label, vertex_name
                        ),
                    ));
                }
            }
            temp_edge_data.insert(edge_label.clone(), (start_node, end_node));
        }
    }

    // Преобразуем собранные данные о дугах в финальный список
    for (label, (u, v)) in temp_edge_data {
        if u == 0 || v == 0 {
            eprintln!("Предупреждение: Дуга '{}' не полностью определена. Игнорируется.", l
            continue;
        }
        edges_list.push((u, v, label));
    }

    Ok((edges_list, temp_edge_data))
}
```

Приложение А2. Исходный код

```rust
use std::fs::File;
use std::io::{self, Write};
use std::collections::HashSet;

pub fn save_dot(edges: &[(usize, usize, String)], filename: &str) -> io::Result<()> {
    let mut dot = String::new();
    dot.push_str("digraph G {\n");
    dot.push_str("  rankdir=LR;\n");
    dot.push_str("  node [shape=circle];\n");

    let nodes: HashSet<usize> = edges.iter()
        .flat_map(|(u, v, _)| vec![*u, *v])
        .collect();
    for node in nodes {
        dot.push_str(&format!("  {};\n", node));
    }

    for (u, v, label) in edges {
        if *u == *v {
            dot.push_str(&format!("  {} -> {} [label=\"{}\"];\n", u, v, label));
        } else {
            dot.push_str(&format!("  {} -> {} [label=\"{}\"];\n", u, v, label));
        }
    }

    dot.push_str("}\n");
    let mut file = File::create(filename)?;
    file.write_all(dot.as_bytes())?;

    println!("Граф сохранён в файл: {}", filename);
    Ok(())
}
```

Приложение А3. Входной файл.

	M	N	P	Q
1	1	-1	0	0
2	1	1	0	0
3	0	0	1	-1
4	0	0	-1	1