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.

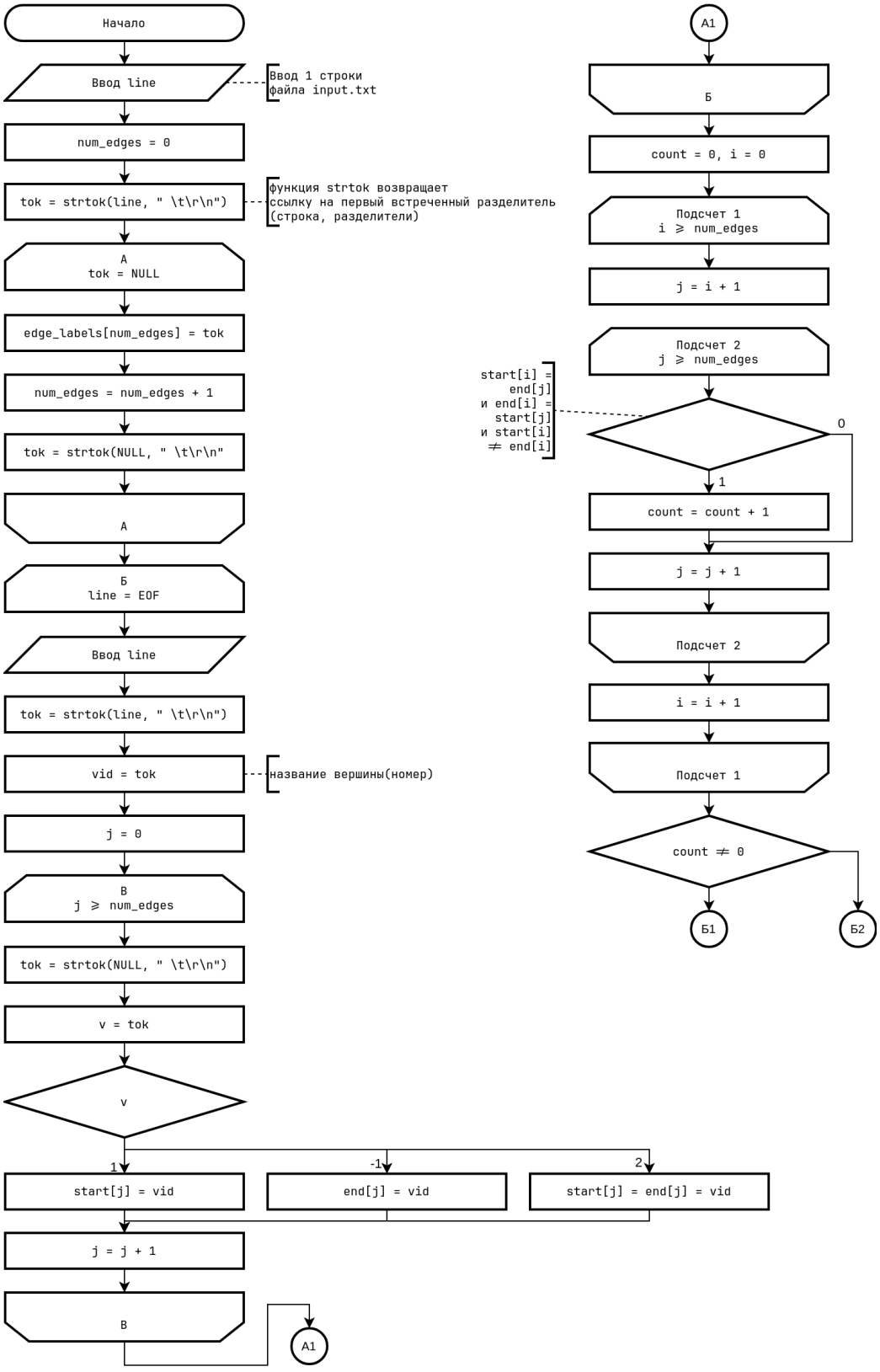

Рисунок 1.1 - Схема алгоритма основной программы стр.1.

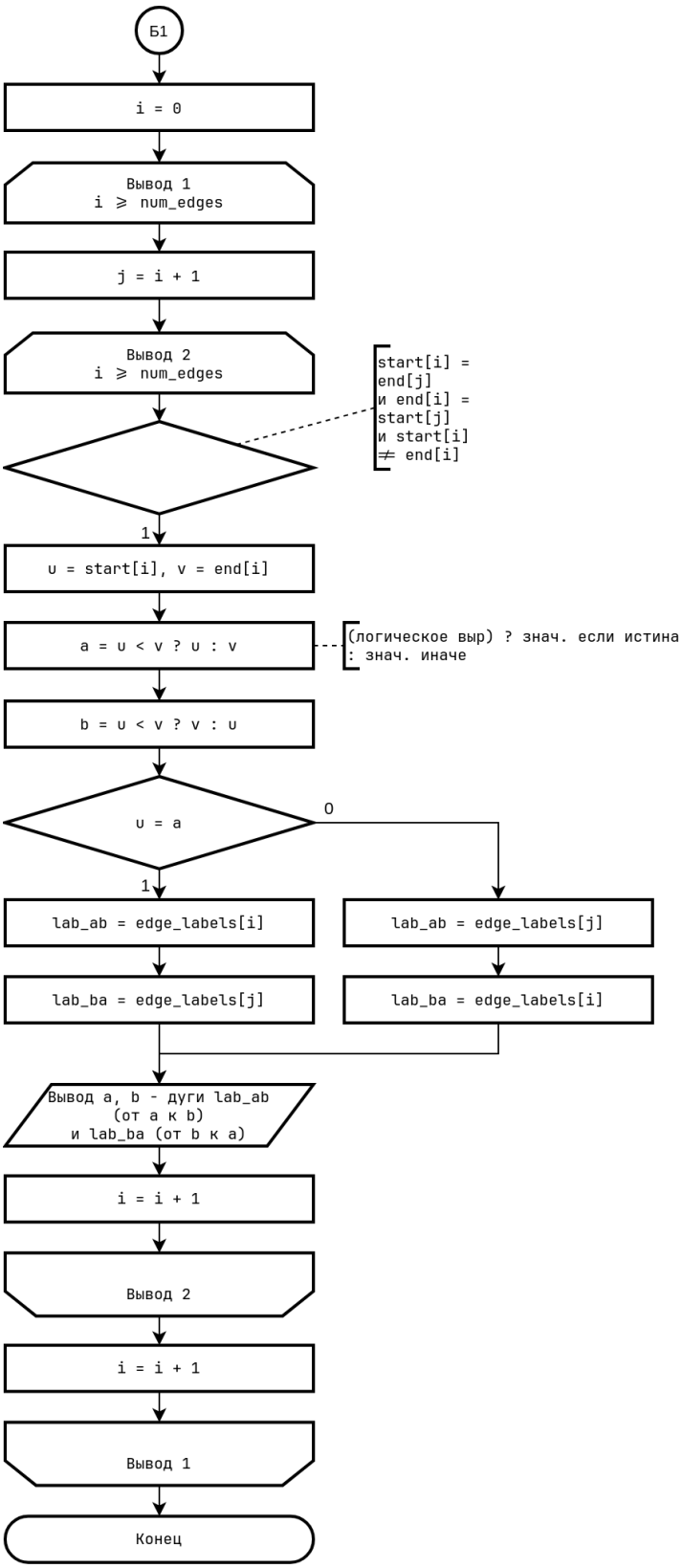

Рисунок 1.2 - Схема алгоритма основной программы стр.2.

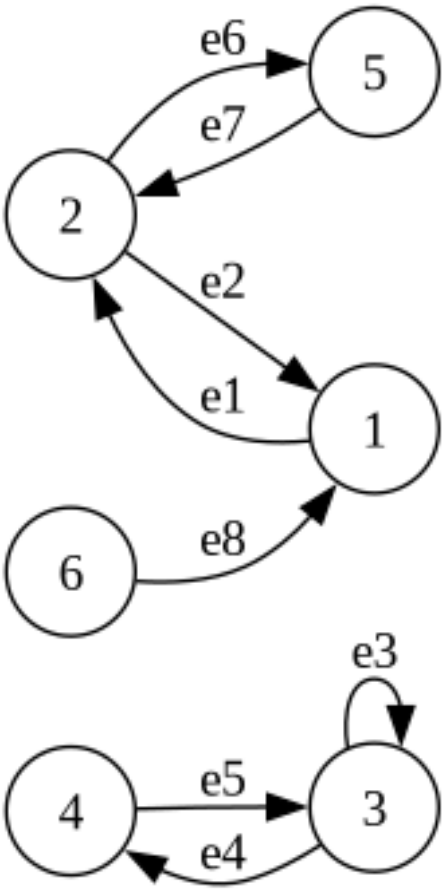

Рисунок 2.2 - Сгенерированный по входному файлу input.txt граф.

```
Содержимое файла "input.txt":
    e1  e2  e3  e4  e5  e6  e7  e8
1  -1   1   0   0   0   0   0   1
2   1  -1   0   0   0  -1   1   0
3   0   0   2  -1   1   0   0   0
4   0   0   0   1  -1   0   0   0
5   0   0   0   0   0   1  -1   0
6   0   0   0   0   0   0   0  -1
──────────────────────────────────────────

Граф сохранён в файл: graph.dot

--- Результаты ---
Количество двунаправленных дуг: 3
Множество найденных дуг:
  (3, 4) – дуги "e4" (от 3 к 4) и "e5" (от 4 к 3)
  (1, 2) – дуги "e1" (от 1 к 2) и "e2" (от 2 к 1)
  (2, 5) – дуги "e7" (от 2 к 5) и "e6" (от 5 к 2)
──────────────────
```

Рисунок 2.1 - Пример работы программы.

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() {
    let enable_visualization = {
        let args: Vec<String> = env::args().collect();
        args.contains(&"-viz".to_string()) || args.contains(&"--visualize".to_string())
    };

    println!("Содержимое файла \"input.txt\":");
    if let Ok(file) = File::open("input.txt") {
        let reader = BufReader::new(file);
        for line_result in reader.lines() {
            if let Ok(line) = line_result {
                println!("{}", line);
            }
        }
    }
    println!("----------------------------------------\n");

    let (edges, _edge_labels) = match read_incidence_matrix("input.txt") {
        Ok(res) => res,
        Err(err) => {
            eprintln!("Ошибка чтения матрицы инцидентности: {}", err);
            return;
        }
    };
```

```rust
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
}

let edge_map: HashMap<(usize, usize), String> = edges
    .iter()
    .map(|(u, v, label)| ((*u, *v), label.clone()))
    .collect();

let mut found_bidirectional_arcs = HashSet::new();
for (u, v, label_uv) in &edges {
    if let Some(label_vu) = edge_map.get(&(*v, *u)) {
        if *u != *v {
            let ordered_pair = if *u < *v { (*u, *v) } else { (*v, *u) };
            found_bidirectional_arcs.insert((ordered_pair, label_uv.clone(), label_vu.c
        }
    }
}

let mut unique_bidirectional = HashMap::new();
for ((a, b), lab1, lab2) in found_bidirectional_arcs {
    unique_bidirectional.entry((a, b)).or_insert((lab1, lab2));
}

println!("\n--- Результаты ---");
println!(
    "Количество двунаправленных дуг: {}",
    unique_bidirectional.len()
);
```

```
    println!("Множество найденных дуг:");
    if unique_bidirectional.is_empty() {
        println!("  Двунаправленные дуги не найдены.");
    } else {
        for ((a, b), (lab_ab, lab_ba)) in unique_bidirectional {
            println!(
                "  ({}, {}) - дуги \"{}\" (от {} к {}) и \"{}\" (от {} к {})",
                a, b, lab_ab, a, b, lab_ba, b, a
            );
        }
    }
    println!("------------------\n");
}

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();

    let header = lines.next().ok_or(io::Error::new(
        io::ErrorKind::InvalidData,
        "Файл пуст или 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_list = Vec::new();
    let mut temp_edge_data: HashMap<String, (usize, usize)> = HashMap::new();
    let mut vertex_name_to_id = HashMap::new();
```

```
let mut next_vertex_id = 1;

for line_result in lines {
    let line = line_result?;
    let parts: Vec<&str> = line.split_whitespace().collect();
    if parts.is_empty() {
        continue;
    }

    let vertex_name = parts[0];
    let current_vertex_id = *vertex_name_to_id
        .entry(vertex_name.to_string())
        .or_insert_with(|| {
            let id = next_vertex_id;
            next_vertex_id += 1;
            id
        });

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

    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 значение '{}' для дуги '{}': {}",
                    value_str, edge_label, e
                ),
            )
        })?;

        let (mut start_node, mut end_node) =
            temp_edge_data.get(edge_label).cloned().unwrap_or((0, 0));

        match incidence_value {
            -1 => {
                // Эта вершина - начало дуги
                start_node = current_vertex_id;
            }
            1 => {
                // Эта вершина - конец дуги
                end_node = current_vertex_id;
            }
            2 => {
                // Петля (начало и конец в одной вершине)
                start_node = current_vertex_id;
                end_node = current_vertex_id;
            }
            0 => {
                // Нет связи, пропуск
            }
            _ => {
                return Err(io::Error::new(
                    io::ErrorKind::InvalidData,
                    format!(
                        "Недопустимое значение '{}' для дуги '{}' и вершины '{}'. Ожида
                        incidence_value, edge_label, vertex_name
```

```
                        ),
                    ));
                }
            }

            temp_edge_data.insert(edge_label.clone(), (start_node, end_node));
        }
    }

    for (label, &(u, v)) in temp_edge_data.iter() {
        if u == 0 || v == 0 {
            eprintln!(
                "Предупреждение: Дуга '{}' не полностью определена. Игнорируется.",
                label
            );
            continue;
        }
        edges_list.push((u, v, label.clone()));
    }

    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.

	e1	e2	e3	e4	e5	e6	e7	e8
1	-1	1	0	0	0	0	1	0
2	1	-1	0	0	0	-1	0	1
3	0	0	2	-1	1	0	0	0
4	0	0	0	1	-1	0	0	0
5	0	0	0	0	0	1	0	-1
6	0	0	0	0	0	0	-1	0