Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №9

з дисципліни
«Алгоритми і структури даних»

Виконав:
студент групи ІМ-42
Туров Андрій Володимирович
номер у списку групи: 28

Перевірив:
Сергієнко А. М.

Київ 2025

Завдання

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:
   (а) кількість вершин n;
   (б) розміщення вершин;
   (в) матриця суміжності A.
2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Варіант 28

$\overline{n_1 n_2 n_3 n_4} = 4228$;
Кількість вершин — $10 + n_3 = 12$.
Розміщення вершин — прямокутником з вершиною в центрі.

Текст програм

```rust
use std::f32::consts::PI;

use raylib::prelude::*;

const FONT_SIZE: i32 = 32;
const CHAR_WIDTH: f32 = 0.27;

const VERTEX_RADIUS: f32 = 20.0;

const ARROWHEAD_LEN: f32 = VERTEX_RADIUS * 0.5;
const ARROWHEAD_ANGLE: f32 = PI / 6.0;

pub fn draw_text(d: &mut RaylibDrawHandle, font: &Font, text:
↪  &str, position: Vector2) {
    d.draw_text_ex(font, text, position, FONT_SIZE as f32, 0.0,
    ↪  Color::BLACK);
}

pub fn draw_vertex(d: &mut RaylibDrawHandle, center: Vector2,
↪  weight: &str, font: &Font) {
    d.draw_circle_lines(center.x as i32, center.y as i32,
    ↪  VERTEX_RADIUS, Color::BLUE);
    if !weight.is_empty() {
        let text_len = weight.chars().count();
        let x_offset: f32 = text_len as f32 * FONT_SIZE as f32 *
        ↪  CHAR_WIDTH;
```

```
        let y_offset: f32 = FONT_SIZE as f32 * 0.5;
        draw_text(
            d,
            font,
            weight,
            Vector2 {
                x: center.x - x_offset,
                y: center.y - y_offset,
            },
        )
    }
}

fn draw_arrowhead(d: &mut RaylibDrawHandle, position: Vector2,
 ↪  direction: Vector2) {
    d.draw_line_v(
        position,
        position - direction.rotated(ARROWHEAD_ANGLE) *
         ↪  ARROWHEAD_LEN,
        Color::BLACK,
    );
    d.draw_line_v(
        position,
        position - direction.rotated(-ARROWHEAD_ANGLE) *
         ↪  ARROWHEAD_LEN,
        Color::BLACK,
    );
}

pub fn draw_straight_edge(
    d: &mut RaylibDrawHandle,
    center_from: Vector2,
    center_to: Vector2,
    directional: bool,
) {
    let direction = (center_to - center_from).normalized();

    let from = center_from + direction * VERTEX_RADIUS;
    let to = center_to - direction * VERTEX_RADIUS;
    d.draw_line_v(from, to, Color::BLACK);
    if directional {
        draw_arrowhead(d, to, direction);
    }
}

pub fn draw_angled_edge(
    d: &mut RaylibDrawHandle,
    center_from: Vector2,
    center_to: Vector2,
    directed: bool,
) {
```

```
    const EDGE_BASE_ANGLE: f32 = 0.05 * PI;

    let direction = (center_to - center_from).normalized();
    let from = center_from + direction * VERTEX_RADIUS;
    let to = center_to - direction * VERTEX_RADIUS;

    let vector = to - from;
    let vector_middle = vector * 0.5;
    let mid_offset = vector_middle.length() *
     ↪  EDGE_BASE_ANGLE.tan();

    let midpoint = from + vector_middle + direction.rotated(0.5 *
     ↪  PI) * mid_offset;
    d.draw_line_v(from, midpoint, Color::BLACK);
    d.draw_line_v(midpoint, to, Color::BLACK);
    if directed {
        draw_arrowhead(d, to, (to - midpoint).normalized());
    }
}

pub fn draw_looping_edge(d: &mut RaylibDrawHandle, center:
 ↪  Vector2) {
    const POINTS: usize = 16;
    const RADIUS: f32 = 12.0;
    const START_ANGLE: f32 = -0.9 * PI;
    const END_ANGLE: f32 = 0.75 * PI;

    let step = (END_ANGLE - START_ANGLE) / POINTS as f32;
    let start_point = center
        + Vector2 {
            x: VERTEX_RADIUS,
            y: -0.85 * VERTEX_RADIUS,
        };

    let points: [Vector2; POINTS] = std::array::from_fn(|i|
     ↪  Vector2 {
        x: start_point.x + RADIUS * f32::cos(START_ANGLE + (step
         ↪  * i as f32)),
        y: start_point.y + RADIUS * f32::sin(START_ANGLE + (step
         ↪  * i as f32)),
    });
    let last_point = points[POINTS - 1];
    d.draw_line_strip(&points, Color::BLACK);

    let direction = (last_point - points[POINTS -
     ↪  4]).normalized();
    d.draw_line_v(
        last_point,
        last_point - direction.rotated(ARROWHEAD_ANGLE) *
         ↪  ARROWHEAD_LEN,
```

```
        Color::BLACK,
    );
    d.draw_line_v(
        last_point,
        last_point - direction.rotated(-ARROWHEAD_ANGLE) *
        ↪  ARROWHEAD_LEN,
        Color::BLACK,
    );
}
```

Файл 1: draw.rs

```
use std::fmt::Display;

use rand::{Rng, SeedableRng, rngs::SmallRng};

// $\overline{n_1n_2n_3n_4} = 4228$
pub const ROWS: &[usize] = &[4, 3, 5];
pub const VERTEX_COUNT: usize = 12; // $10 + n_3 = 12$
const RANDOM_SEED: u64 = 4228;

#[derive(Clone)]
pub struct AdjMatrix(pub [[u8; VERTEX_COUNT]; VERTEX_COUNT]);

impl Display for AdjMatrix {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) ->
    ↪  std::fmt::Result {
        for i in 0..VERTEX_COUNT {
            for j in 0..VERTEX_COUNT {
                if i == j {
                    write!(f, "\x1b[31m{}\x1b[0m ",
                    ↪  self.0[i][j])?;
                    continue;
                }
                write!(f, "{} ", self.0[i][j])?;
            }
            writeln!(f)?;
        }
        Ok(())
    }
}

pub fn generate_dir_matrix() -> AdjMatrix {
    let mut rng = SmallRng::seed_from_u64(RANDOM_SEED);
    let iter = std::iter::repeat_with(move ||
    ↪  rng.random_range(0.0..2.0));

    // $1 - n_3 * 0.02 - n_4 * 0.005 - 0.25$
    const K: f32 = 1.0 - 2.0 * 0.02 - 8.0 * 0.005 - 0.25;
```

```
    let mut values = iter.take(VERTEX_COUNT * VERTEX_COUNT);
    let matrix = std::array::from_fn(|_| std::array::from_fn(|_|
    ↪  values.next().unwrap() * K));

    AdjMatrix(matrix.map(|row| row.map(|value| f32::min(1.0,
    ↪  value) as u8)))
}

pub fn convert_to_undir(dir_matrix: &AdjMatrix) -> AdjMatrix {
    let mut undir_matrix = dir_matrix.clone();
    for i in 0..VERTEX_COUNT {
        for j in (i + 1)..VERTEX_COUNT {
            undir_matrix.0[j][i] = undir_matrix.0[i][j];
        }
    }
    undir_matrix
}
```

Файл 2: graph.rs

```
use draw::draw_text;
use graph::{AdjMatrix, ROWS, VERTEX_COUNT};
use raylib::{color::Color, prelude::*};
mod draw;
mod graph;

const WIN_WIDTH: i32 = 800;
const WIN_HEIGHT: i32 = 600;
const WIN_MARGIN: f32 = 0.8;

#[derive(Clone, Copy)]
struct VertexPos {
    v: Vector2,
    row: usize,
    col: usize,
}

fn draw_all_vertices(d: &mut RaylibDrawHandle, font: &Font) ->
↪  [VertexPos; VERTEX_COUNT] {
    fn current_position(index: usize) -> (usize, usize) {
        let mut cumulative = 0;
        for (row, &count) in ROWS.iter().enumerate() {
            if index < cumulative + count {
                return (row, index - cumulative);
            }
            cumulative += count;
        }
        (usize::MAX, usize::MAX)
    }
    let winwidth = WIN_WIDTH as f32 * WIN_MARGIN;
```

```
    let winheight = WIN_HEIGHT as f32 * WIN_MARGIN;
    let vertex_coords: [VertexPos; VERTEX_COUNT] =
     ↪  std::array::from_fn(|i| {
        let (row, col) = current_position(i);

        let x_offset = (WIN_WIDTH as f32 - winwidth) * 0.5;
        let y_offset = (WIN_HEIGHT as f32 - winheight) * 0.5;
        VertexPos {
            v: Vector2 {
                x: (winwidth / (ROWS[row] - 1) as f32 * col as
                 ↪  f32) + x_offset,
                y: (winheight / (ROWS.len() - 1) as f32 * row as
                 ↪  f32) + y_offset,
            },
            row,
            col,
        }
    });

    (0..VERTEX_COUNT).for_each(|i| {
        draw::draw_vertex(d, vertex_coords[i].v, &((i +
         ↪  1).to_string()), font);
    });

    vertex_coords
}

fn draw_all_edges(
    d: &mut RaylibDrawHandle,
    adj_matrix: &AdjMatrix,
    vertex_coords: &[VertexPos],
    directed: bool,
) {
    (0..VERTEX_COUNT).for_each(|i| {
        let lower = if directed { 0 } else { i };
        (lower..VERTEX_COUNT).for_each(|j| {
            let origin = vertex_coords[i];
            let destination = vertex_coords[j];
            let row_absdiff = (destination.row as i64 -
             ↪  origin.row as i64).abs();
            let col_absdiff = (destination.col as i64 -
             ↪  origin.col as i64).abs();

            if adj_matrix.0[i][j] == 1 {
                if i == j {
                    draw::draw_looping_edge(d, origin.v);
                } else if (adj_matrix.0[j][i] == 1 && directed)
                 ↪  // symmetric
                    || (row_absdiff == 0 && col_absdiff > 1) //
                     ↪  same row, goes through others
```

```
                    || (col_absdiff == 0 && row_absdiff > 1) //
                    ↪  same col, goes through others
                    || (origin.v.x == destination.v.x) // same x
                    ↪  coordinate, yes, still possible
                    || col_absdiff ⩾ 3
                // honestly ^ whatever this is
                {
                    draw::draw_angled_edge(d, origin.v,
                    ↪  destination.v, directed);
                } else {
                    draw::draw_straight_edge(d, origin.v,
                    ↪  destination.v, directed);
                }
            }
        })
    })
}

fn main() {
    let dir_matrix = graph::generate_dir_matrix();
    let undir_matrix = graph::convert_to_undir(&dir_matrix);
    println!("Directed adjacency matrix:\n{}", dir_matrix);
    println!("Undirected adjacency matrix:\n{}", undir_matrix);

    let (mut rl, thread) = raylib::init()
        .size(WIN_WIDTH, WIN_HEIGHT)
        .log_level(TraceLogLevel::LOG_WARNING)
        .title("ASD Lab 2.3")
        .build();

    let font = rl.load_font(&thread,
    ↪  "FiraCode-Regular.ttf").unwrap();

    while !rl.window_should_close() {
        let mut d = rl.begin_drawing(&thread);

        d.clear_background(Color::WHITE);
        draw_text(
            &mut d,
            &font,
            "<Space>",
            Vector2 {
                x: 0.8 * WIN_WIDTH as f32,
                y: 0.95 * WIN_HEIGHT as f32,
            },
        );

        let vertex_coords = draw_all_vertices(&mut d, &font);
        if d.is_key_down(KeyboardKey::KEY_SPACE) {
```

```
            draw_all_edges(&mut d, &dir_matrix, &vertex_coords,
             ↪  true);
        } else {
            draw_all_edges(&mut d, &undir_matrix, &vertex_coords,
             ↪  false);
        }
    }
}
```

Файл 3: main.rs

Матриці суміжності

Для напрямленого графа:

```
0 1 1 0 0 0 1 0 0 0 1 0
0 0 0 1 0 0 0 1 0 0 1 1
0 0 1 0 0 1 1 0 0 0 0 0
0 0 1 1 0 1 0 1 1 1 0 1
0 0 0 0 0 0 1 0 0 0 1 0
0 0 0 0 0 1 0 1 0 1 0 0
0 0 1 0 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 1 1
1 0 1 1 0 1 0 0 1 1 0 1
```

Для ненапрямленого графа:

```
0 1 1 0 0 0 1 0 0 0 1 0
1 0 0 1 0 0 0 1 0 0 1 1
1 0 1 0 0 1 1 0 0 0 0 0
0 1 0 1 0 1 0 1 1 1 0 1
0 0 0 0 0 0 1 0 0 0 1 0
0 0 1 1 0 1 0 1 0 1 0 0
1 0 1 0 1 0 0 0 0 0 1 0
0 1 0 1 0 1 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 1 0 1 0 1 0 0 0 0
1 1 0 0 1 0 1 0 1 0 1 1
0 1 0 1 0 0 0 0 0 0 1 1
```

Зображення

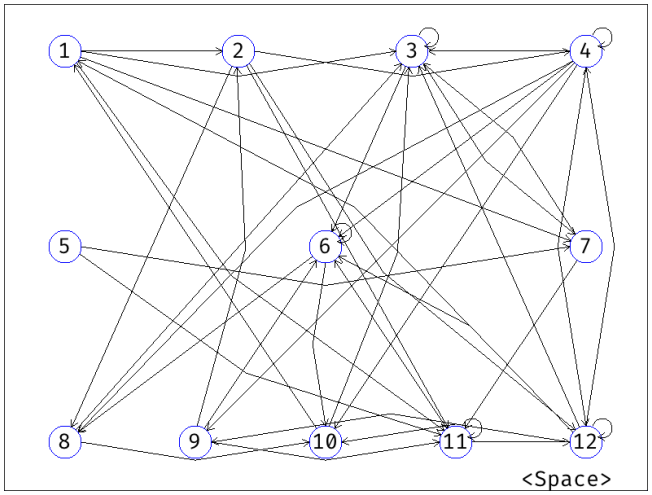

Рис. 1: Напрямлений граф

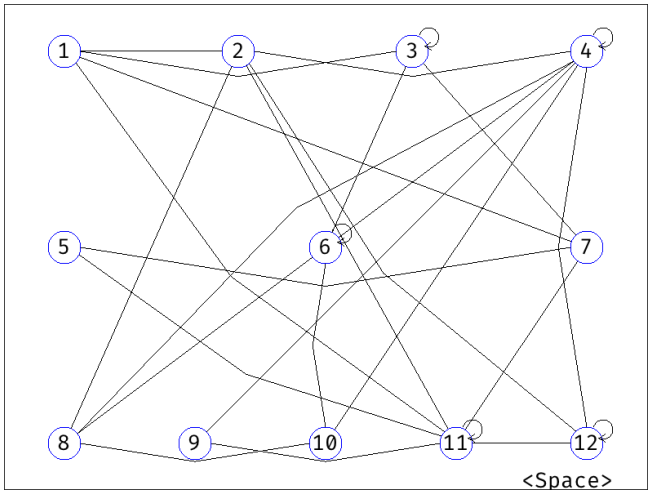

Рис. 2: Ненапрямлений граф

Висновок

Використав Raylib мовою програмування Rust для зображення графа за заданою матрицею суміжності.
За допомогою графічних примітивів намалював вершини та ребра, алгоритмічно вибираючи між прямим сполученням між колами та сполученням ламаною, що проходить (у звіті) під основним кутом $\pi/20$.
У програмі граф повністю представлений своєю матрицею суміжності, фактичною та відносною позицією вершин у просторі.