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 №10

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 напрямлений та ненапрямлений графи із заданими параметрами так само, як у лабораторній роботі №3.

   Відмінність: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$.

2. Обчислити:

   (а) степені вершин напрямленого і ненапрямленого графів;

   (б) напівстепені виходу та заходу напрямленого графа;

   (в) чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;

   (г) перелік висячих та ізольованих вершин.

3. Змінити матрицю A_{dir}, коефіцієнт $k = 1.0 * n_3 * 0.005 * n_4 * 0.005 * 0.27$.

4. Для нового орграфа обчислити:

   (а) півстепені вершин;

   (б) всі шляхи довжини 2 і 3;

   (в) матрицю досяжності;

   (г) матрицю сильної зв'язності;

   (д) перелік компонент сильної зв'язності;

   (е) граф конденсації.

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::{
    collections::HashMap,
    fmt::Display,
    ops::{Add, AddAssign, BitAnd, BitAndAssign, BitOr,
     ↪  BitOrAssign, Mul, MulAssign},
    vec,
};

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

// $\overline{n_1n_2n_3n_4} = 4228$
pub const DEFAULT_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 Vec<Vec<u32>>);

impl Display for AdjMatrix {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) ->
     ↪  std::fmt::Result {
```

```
        if f.alternate() {
            let degrees = (0..self.0.len()).map(|i|
             ↪  self.degree(i));
            let is_regular = self.is_graph_regular();
            let isolated = (0..self.0.len()).filter(|i|
             ↪  self.is_isolated(*i));
            let pendant = (0..self.0.len()).filter(|i|
             ↪  self.is_pendant(*i));

            write!(f, "{:<27}", "Vertex degrees:")?;
            degrees.for_each(|deg| write!(f, "{} ", deg).map(|_|
             ↪  ()).unwrap());
            writeln!(f, "\nIs regular: {}", is_regular)?;
            if is_regular {
                writeln!(f, "Regularity degree: {}",
                 ↪  self.degree(0))?;
            }
            write!(f, "{:<27}", "Isolated vertices:")?;
            isolated.for_each(|vertex| write!(f, "{} ", vertex +
             ↪  1).map(|_| ()).unwrap());
            write!(f, "\n{:<27}", "Pendant vertices:")?;
            pendant.for_each(|vertex| write!(f, "{} ", vertex +
             ↪  1).map(|_| ()).unwrap());
        } else {
            for i in 0..self.0.len() {
                for j in 0..self.0.len() {
                    if i == j {
                        write!(f, "\x1b[31m{}\x1b[0m ",
                         ↪  self.0[i][j])?;
                        continue;
                    }
                    write!(f, "{} ", self.0[i][j])?;
                }
                writeln!(f)?;
            }
        }
        Ok(())
    }
}

impl<'a> Mul<&'a AdjMatrix> for &AdjMatrix {
    type Output = AdjMatrix;

    fn mul(self, rhs: &'a AdjMatrix) -> Self::Output {
        let mut result = vec![vec![0_u32; self.0.len()];
         ↪  self.0.len()];

        for i in 0..self.0.len() {
            for j in 0..self.0.len() {
                let mut sum = 0_u32;
```

```
                for k in 0..self.0.len() {
                    sum += self.0[i][k] * rhs.0[k][j];
                }
                result[i][j] = sum;
            }
        }

        AdjMatrix(result)
    }
}

impl MulAssign<&AdjMatrix> for AdjMatrix {
    fn mul_assign(&mut self, rhs: &AdjMatrix) {
        *self = &*self * rhs;
    }
}

impl<'a> Add<&'a AdjMatrix> for &AdjMatrix {
    type Output = AdjMatrix;

    fn add(self, rhs: &'a AdjMatrix) -> Self::Output {
        AdjMatrix(
            self.0
                .iter()
                .zip(&rhs.0)
                .map(|(row_a, row_b)|
                ↪  row_a.iter().zip(row_b.iter()).map(|(a, b)| a
                ↪  + b).collect())
                .collect(),
        )
    }
}

impl AddAssign<&AdjMatrix> for AdjMatrix {
    fn add_assign(&mut self, rhs: &AdjMatrix) {
        *self = &*self + rhs;
    }
}

impl<'a> BitOr<&'a AdjMatrix> for &AdjMatrix {
    type Output = AdjMatrix;

    fn bitor(self, rhs: &'a AdjMatrix) -> Self::Output {
        AdjMatrix(
            self.0
                .iter()
                .zip(&rhs.0)
                .map(|(row_a, row_b)| {
                    row_a
                        .iter()
```

```
                        .zip(row_b.iter())
                        .map(|(a, b)| (*a ≠ 0 || *b ≠ 0) as
                        ↪  u32)
                        .collect()
                })
                .collect(),
        )
    }
}

impl BitOrAssign<&AdjMatrix> for AdjMatrix {
    fn bitor_assign(&mut self, rhs: &AdjMatrix) {
        *self = &*self | rhs;
    }
}

impl<'a> BitAnd<&'a AdjMatrix> for &AdjMatrix {
    type Output = AdjMatrix;

    fn bitand(self, rhs: &'a AdjMatrix) -> Self::Output {
        AdjMatrix(
            self.0
                .iter()
                .zip(&rhs.0)
                .map(|(row_a, row_b)| {
                    row_a
                        .iter()
                        .zip(row_b.iter())
                        .map(|(a, b)| (*a ≠ 0 && *b ≠ 0) as
                        ↪  u32)
                        .collect()
                })
                .collect(),
        )
    }
}

impl BitAndAssign<&AdjMatrix> for AdjMatrix {
    fn bitand_assign(&mut self, rhs: &AdjMatrix) {
        *self = &*self & rhs;
    }
}

impl AdjMatrix {
    pub fn identity(size: usize) -> Self {
        AdjMatrix(
            (0..size)
                .map(|i| (0..size).map(|j| if i == j { 1 } else {
                ↪  0 }).collect())
                .collect(),
```

```
        )
    }
    pub fn transpose(&self) -> Self {
        AdjMatrix(
            (0..self.0.len())
                .map(|col| self.0.iter().map(|row|
                ↪  row[col]).collect())
                .collect(),
        )
    }
}

impl AdjMatrix {
    pub fn generate(k: f32) -> Self {
        let mut rng = SmallRng::seed_from_u64(RANDOM_SEED);
        let iter = std::iter::repeat_with(move ||
        ↪  rng.random_range(0.0..2.0));

        AdjMatrix(
            iter.take(VERTEX_COUNT * VERTEX_COUNT)
                .map(|i| f32::min(i * k, 1.0) as u32)
                .collect::<Vec<_>>()
                .chunks(VERTEX_COUNT)
                .map(|row| row.to_vec())
                .collect(),
        )
    }

    pub fn undir(&self) -> Self {
        let mut undir_matrix = self.clone();
        for i in 0..self.0.len() {
            for j in (i + 1)..self.0.len() {
                undir_matrix.0[j][i] = undir_matrix.0[i][j];
            }
        }
        undir_matrix
    }

    pub fn degree_out(&self, vertex: usize) -> usize {
        self.0[vertex].iter().filter(|i| **i == 1).count()
    }

    pub fn degree_in(&self, vertex: usize) -> usize {
        self.0
            .iter()
            .map(|row| row[vertex])
            .filter(|i| *i == 1)
            .count()
    }
```

```
pub fn degree(&self, vertex: usize) -> usize {
    self.degree_out(vertex) + self.degree_in(vertex)
}

pub fn is_graph_regular(&self) -> bool {
    let mut degrees = (0..self.0.len()).map(|vertex|
    ↪  self.degree(vertex));
    if let Some(first) = degrees.next() {
        degrees.all(|deg| deg == first)
    } else {
        true
    }
}

pub fn is_isolated(&self, vertex: usize) -> bool {
    self.degree(vertex) == 0
}

pub fn is_pendant(&self, vertex: usize) -> bool {
    self.degree(vertex) == 1
}

pub fn all_paths_of_2(&self) -> Vec<[usize; 3]> {
    let squared = self * self;
    let capacity: usize =
    ↪  squared.0.iter().flatten().sum::<u32>() as usize;
    let mut paths = Vec::with_capacity(capacity);

    for i in 0..self.0.len() {
        for j in 0..self.0.len() {
            let mut remaining = squared.0[i][j];
            if remaining > 0 {
                for k in 0..self.0.len() {
                    if self.0[i][k] == 1 && self.0[k][j] == 1
                    ↪  {
                        paths.push((i, k, j).into());
                        remaining -= 1;
                    }
                }
            }
        }
    }
    paths
}

pub fn all_paths_of_3(&self) -> Vec<[usize; 4]> {
    let cubed = &(self * self) * self;
    let capacity = cubed.0.iter().flatten().sum::<u32>() as
    ↪  usize;
    let mut paths = Vec::with_capacity(capacity);
```

```
        for i in 0..self.0.len() {
            for j in 0..self.0.len() {
                let mut remaining = cubed.0[i][j];
                if remaining > 0 {
                    for k in 0..self.0.len() {
                        for l in 0..self.0.len() {
                            if self.0[i][k] == 1 && self.0[k][l]
                            ↪  == 1 && self.0[l][j] == 1 {
                                paths.push((i, k, l, j).into());
                                remaining -= 1;
                            }
                        }
                    }
                }
            }
        }
        paths
    }

    pub fn reachability(&self) -> Self {
        let mut result = Self::identity(self.0.len());
        let mut next_power = self.clone();
        for _ in 1..self.0.len() {
            result |= &next_power;
            next_power *= self;
        }
        if self.0.len() > 1 {
            result |= &next_power;
        }
        result
    }

    pub fn connectivity(&self) -> Self {
        let reach = self.reachability();
        &reach & &reach.transpose()
    }

    pub fn conn_components(&self) -> Vec<Vec<usize>> {
        let mut result: Vec<Vec<usize>> =
        ↪  Vec::with_capacity(self.0.len());
        let mut lookup: HashMap<Vec<u32>, usize> =
        ↪  HashMap::with_capacity(self.0.len());
        let conn = self.connectivity();

        for (i, row) in conn.0.iter().enumerate() {
            if let Some(group) = lookup.get(row) {
                result[*group].push(i);
            } else {
                result.push(Vec::with_capacity(self.0.len()));
```

```
                let index = result.len() - 1;
                lookup.insert(row.to_vec(), index);
                result[index].push(i);
            }
        }

        result
    }

    pub fn condensed(&self) -> Self {
        fn comp_index(components: &[Vec<usize>], vertex: usize)
         ↪  -> std::option::Option<usize> {
            components.iter().position(|comp|
             ↪  comp.contains(&vertex))
        }
        let components = self.conn_components();
        let mut cond_matrix = vec![vec![0; components.len()];
         ↪  components.len()];
        for i in 0..self.0.len() {
            for j in 0..self.0.len() {
                if self.0[i][j] == 1 {
                    let comp_i = comp_index(&components,
                     ↪  i).unwrap();
                    let comp_j = comp_index(&components,
                     ↪  j).unwrap();
                    if comp_i ≠ comp_j {
                        cond_matrix[comp_i][comp_j] = 1;
                    }
                }
            }
        }

        AdjMatrix(cond_matrix)
    }
}
```

Файл 2: graph.rs

```
#![allow(clippy::needless_range_loop)]

use graph::{AdjMatrix, DEFAULT_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;
```

```rust
const K_1: f32 = 1.0 - 2.0 * 0.01 - 8.0 * 0.01 - 0.3;
const K_2: f32 = 1.0 - 2.0 * 0.005 - 8.0 * 0.005 - 0.27;

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

fn draw_all_vertices(d: &mut RaylibDrawHandle, font: &Font, rows:
 ↪  &[usize]) -> Vec<VertexPos> {
    fn current_position(index: usize, rows: &[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_count = rows.iter().sum();
    let vertex_coords: Vec<VertexPos> =
     ↪  Vec::from_iter((0..vertex_count).map(|i| {
        let (row, col) = current_position(i, rows);

        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 / (DEFAULT_ROWS[row] - 1) as f32 *
                 ↪  col as f32) + x_offset,
                y: (winheight / (DEFAULT_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,
) {
    for i in 0..vertex_coords.len() {
        let lower = if directed { 0 } else { i };
        for j in lower..vertex_coords.len() {
            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 generate_and_print() -> (AdjMatrix, AdjMatrix, AdjMatrix,
↪  AdjMatrix) {
    let dir_matrix = AdjMatrix::generate(K_1);
    let in_degrees = (0..VERTEX_COUNT).map(|i|
    ↪  dir_matrix.degree_in(i));
    let out_degrees = (0..VERTEX_COUNT).map(|i|
    ↪  dir_matrix.degree_out(i));
```

```
println!("Directed adjacency matrix:\n{}", dir_matrix);
print!("Vertex semi-degrees (IN):  ");
in_degrees.for_each(|deg| print!("{} ", deg));
print!("\nVertex semi-degrees (OUT): ");
out_degrees.for_each(|deg| print!("{} ", deg));
println!("\n{:#}", dir_matrix);

let undir_matrix = dir_matrix.undir();
println!("\n\n\nUndirected adjacency matrix:\n{}",
 ↪  undir_matrix);
println!("{:#}", undir_matrix);

let dir_matrix2 = AdjMatrix::generate(K_2);
let in_degrees2 = (0..VERTEX_COUNT).map(|i|
 ↪  dir_matrix2.degree_in(i));
let out_degrees2 = (0..VERTEX_COUNT).map(|i|
 ↪  dir_matrix2.degree_out(i));
let dir_paths2: Vec<_> = dir_matrix2
    .all_paths_of_2()
    .iter()
    .map(|path| (path[0] + 1, path[1] + 1, path[2] + 1))
    .map(|(i, k, j)| format!("{}->{}->{}", i, k, j))
    .collect();
let dir_paths3: Vec<_> = dir_matrix2
    .all_paths_of_3()
    .iter()
    .map(|path| (path[0] + 1, path[1] + 1, path[2] + 1,
     ↪  path[3] + 1))
    .map(|(i, k, l, j)| format!("{}->{}->{}->{}", i, k, l,
     ↪  j))
    .collect();
let dir_reach = dir_matrix2.reachability();
let condensed = dir_matrix2.condensed();

println!("\n\n\nModified adjacency matrix:\n{}",
 ↪  dir_matrix2);
print!("Vertex semi-degrees (IN):  ");
in_degrees2.for_each(|deg| print!("{} ", deg));
print!("\nVertex semi-degrees (OUT): ");
out_degrees2.for_each(|deg| print!("{} ", deg));
print!("\nPaths of 2: ");
dir_paths2.iter().for_each(|path| print!("{} ", path));
print!("\n\nPaths of 3: ");
dir_paths3.iter().for_each(|path| print!("{} ", path));
println!("\n\nReachability matrix:\n{}", dir_reach);
println!("Connectivity matrix:\n{}",
 ↪  dir_matrix2.connectivity());
println!(
    "Connectivity components: {:?}",
```

```
        dir_matrix2
            .conn_components()
            .iter()
            .map(|comp| comp.iter().map(|i| i +
             ↪  1).collect::<Vec<_>>())
            .collect::<Vec<_>>()
    );
    println!("Condensed adjacency matrix:\n{}", condensed);

    (dir_matrix, undir_matrix, dir_matrix2, condensed)
}

fn main() {
    let (mut rl, thread) = raylib::init()
        .size(WIN_WIDTH, WIN_HEIGHT)
        .log_level(TraceLogLevel::LOG_WARNING)
        .title("ASD Lab 2.4")
        .build();

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

    let mut state: KeyboardKey = KeyboardKey::KEY_F1;

    let (dir_matrix, undir_matrix, dir_matrix2, condensed) =
     ↪  generate_and_print();

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

        d.clear_background(Color::WHITE);
        if let Some(key) = pressed {
            if [
                KeyboardKey::KEY_F1,
                KeyboardKey::KEY_F2,
                KeyboardKey::KEY_F3,
                KeyboardKey::KEY_F4,
            ]
            .contains(&key)
            {
                state = key;
            }
        }

        draw::draw_text(
            &mut d,
            &font,
            "<F1>    <F2>    <F3>    <F4>",
            Vector2 {
```

```
                x: 0.2 * WIN_WIDTH as f32,
                y: 0.01 * WIN_HEIGHT as f32,
            },
        );

        match state {
            KeyboardKey::KEY_F1 => {
                let vertex_coords = draw_all_vertices(&mut d,
                ↪  &font, DEFAULT_ROWS);
                draw_all_edges(&mut d, &dir_matrix,
                ↪  &vertex_coords, true);
            }
            KeyboardKey::KEY_F2 => {
                let vertex_coords = draw_all_vertices(&mut d,
                ↪  &font, DEFAULT_ROWS);
                draw_all_edges(&mut d, &undir_matrix,
                ↪  &vertex_coords, false);
            }
            KeyboardKey::KEY_F3 => {
                let vertex_coords = draw_all_vertices(&mut d,
                ↪  &font, DEFAULT_ROWS);
                draw_all_edges(&mut d, &dir_matrix2,
                ↪  &vertex_coords, true);
            }
            KeyboardKey::KEY_F4 => {
                let vertex_coords = draw_all_vertices(&mut d,
                ↪  &font, &[condensed.0.len()]);
                draw_all_edges(&mut d, &condensed,
                ↪  &vertex_coords, true);
            }
            _ => {}
        }
    }
}
```

Файл 3: main.rs

Матриці суміжності

Для напрямленого графа:

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

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
```

Характеристика вершин

Для напрямленого графа:

```
Vertex semi-degrees (IN): 1 0 3 3 0 6 1 2 1 0 5 1
Vertex semi-degrees (OUT): 2 2 1 4 2 2 1 0 2 2 2 3
Vertex degrees: 3 2 4 7 2 8 2 2 3 2 7 4
Is regular: false
Isolated vertices:
Pendant vertices:
```

Для ненапрямленого графа:

```
Vertex degrees: 4 4 4 8 4 8 4 4 2 0 10 2
Is regular: false
Isolated vertices: 10
Pendant vertices:
```

Модифікований граф

Матриця суміжності:

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

Напівстепені вершин:

```
vertex semi-degrees (in): 2 2 7
4 0 6 5 3 3 4 6 5
vertex semi-degrees (out): 5 4 3
7 2 3 2 2 5 3 3 8
```

Усі шляхи довжини 2:

```
Paths of 2: 1→3→3 1→4→3 1→7→3 1→2→4 1→4→4 1→3→6 1→4→6
1→11→6 1→3→7 1→2→8 1→4→8 1→4→9 1→4→10 1→2→11 1→7→11
1→11→11 1→2→12 1→4→12 1→11→12 2→12→1 2→4→3 2→8→3 2→12→3
2→4→4 2→12→4 2→4→6 2→11→6 2→12→6 2→12→7 2→4→8 2→4→9
2→12→9 2→4→10 2→8→10 2→12→10 2→11→11 2→4→12 2→11→12 2→12→12
3→3→3 3→7→3 3→3→6 3→6→6 3→3→7 3→6→8 3→6→10 3→7→11 4→10→1
4→12→1 4→9→2 4→3→3 4→4→3 4→8→3 4→10→3 4→12→3 4→4→4 4→12→4
4→3→6 4→4→6 4→6→6 4→9→6 4→12→6 4→3→7 4→10→7 4→12→7 4→4→8
4→6→8 4→4→9 4→9→9 4→12→9 4→4→10 4→6→10 4→8→10 4→12→10
4→9→11 4→4→12 4→9→12 4→12→12 5→7→3 5→11→6 5→7→11 5→11→11
5→11→12 6→10→1 6→8→3 6→10→3 6→6→6 6→10→7 6→6→8 6→6→10
6→8→10 7→3→3 7→3→6 7→11→6 7→3→7 7→11→11 7→11→12 8→10→1
8→3→3 8→10→3 8→3→6 8→3→7 8→10→7 9→12→1 9→9→2 9→12→3 9→2→4
9→12→4 9→6→6 9→9→6 9→11→6 9→12→6 9→12→7 9→2→8 9→6→8 9→9→9
9→12→9 9→6→10 9→12→10 9→2→11 9→9→11 9→11→11 9→2→12 9→9→12
9→11→12 9→12→12 10→1→2 10→1→3 10→3→3 10→7→3 10→1→4 10→3→6
10→1→7 10→3→7 10→1→11 10→7→11 11→12→1 11→12→3 11→12→4
11→6→6 11→11→6 11→12→6 11→12→7 11→6→8 11→12→9 11→6→10
11→12→10 11→11→11 11→11→12 11→12→12 12→10→1 12→12→1 12→1→2
12→9→2 12→1→3 12→3→3 12→4→3 12→7→3 12→10→3 12→12→3 12→1→4
12→4→4 12→12→4 12→3→6 12→4→6 12→6→6 12→9→6 12→12→6 12→1→7
12→3→7 12→10→7 12→12→7 12→4→8 12→6→8 12→4→9 12→9→9 12→12→9
12→4→10 12→6→10 12→12→10 12→1→11 12→7→11 12→9→11 12→4→12
12→9→12 12→12→12
```

Усі шляхи довжини 3:

```
Paths of 3: 1→2→12→1 1→4→10→1 1→4→12→1 1→11→12→1 1→4→9→2
1→2→4→3 1→2→8→3 1→2→12→3 1→3→3→3 1→3→7→3 1→4→3→3
1→4→4→3 1→4→8→3 1→4→10→3 1→4→12→3 1→7→3→3 1→11→12→3
1→2→4→4 1→2→12→4 1→4→4→4 1→4→12→4 1→11→12→4 1→2→4→6
1→2→11→6 1→2→12→6 1→3→3→6 1→3→6→6 1→4→3→6 1→4→4→6
1→4→6→6 1→4→9→6 1→4→12→6 1→7→3→6 1→7→11→6 1→11→6→6
1→11→11→6 1→11→12→6 1→2→12→7 1→3→3→7 1→4→3→7 1→4→10→7
1→4→12→7 1→7→3→7 1→11→12→7 1→2→4→8 1→3→6→8 1→4→4→8
1→4→6→8 1→11→6→8 1→2→4→9 1→2→12→9 1→4→4→9 1→4→9→9
1→4→12→9 1→11→12→9 1→2→4→10 1→2→8→10 1→2→12→10 1→3→6→10
```

1→4→4→10 1→4→6→10 1→4→8→10 1→4→12→10 1→11→6→10 1→11→12→10
1→2→11→11 1→3→7→11 1→4→9→11 1→7→11→11 1→11→11→11 1→2→4→12
1→2→11→12 1→2→12→12 1→4→4→12 1→4→9→12 1→4→12→12 1→7→11→12
1→11→11→12 1→11→12→12 2→4→10→1 2→4→12→1 2→8→10→1
2→11→12→1 2→12→10→1 2→12→12→1 2→4→9→2 2→12→1→2 2→12→9→2
2→4→3→3 2→4→4→3 2→4→8→3 2→4→10→3 2→4→12→3 2→8→3→3
2→8→10→3 2→11→12→3 2→12→1→3 2→12→3→3 2→12→4→3 2→12→7→3
2→12→10→3 2→12→12→3 2→4→4→4 2→4→12→4 2→11→12→4 2→12→1→4
2→12→4→4 2→12→12→4 2→4→3→6 2→4→4→6 2→4→6→6 2→4→9→6
2→4→12→6 2→8→3→6 2→11→6→6 2→11→11→6 2→11→12→6 2→12→3→6
2→12→4→6 2→12→6→6 2→12→9→6 2→12→12→6 2→4→3→7 2→4→10→7
2→4→12→7 2→8→3→7 2-h8→10→7 2→11→12→7 2→12→1→7 2→12→3→7
2→12→10→7 2→12→12→7 2→4→4→8 2→4→6→8 2→11→6→8 2→12→4→8
2→12→6→8 2→4→4→9 2→4→9→9 2→4→12→9 2→11→12→9 2→12→4→9
2→12→9→9 2→12→12→9 2→4→4→10 2→4→6→10 2→4→8→10 2→4→12→10
2→11→6→10 2→11→12→10 2→12→4→10 2→12→6→10 2→12→12→10
2→4→9→11 2→11→11→11 2→12→1→11 2→12→7→11 2→12→9→11
2→4→4→12 2→4→9→12 2→4→12→12 2→11→11→12 2→11→12→12
2→12→4→12 2→12→9→12 2→12→12→12 3→6→10→1 3→3→3→3 3→3→7→3
3→6→8→3 3→6→10→3 3→7→3→3 3→3→3→6 3→3→6→6 3→6→6→6
3→7→3→6 3→7→11→6 3→3→3→7 3→6→10→7 3→7→3→7 3→3→6→8
3→6→6→8 3→3→6→10 3→6→6→10 3→6→8→10 3→3→7→11 3→7→11→11
3→7→11→12 4→4→10→1 4→4→12→1 4→6→10→1 4→8→10→1 4→9→12→1
4→12→10→1 4→12→12→1 4→4→9→2 4→9→9→2 4→10→1→2 4→12→1→2
4→12→9→2 4→3→3→3 4→3→7→3 4→4→3→3 4→4→4→3 4→4→8→3
4→4→10→3 4→4→12→3 4→6→8→3 4→6→10→3 4→8→3→3 4→8→10→3
4→9→12→3 4→10→1→3 4→10→3→3 4→10→7→3 4→12→1→3 4→12→3→3
4→12→4→3 4→12→7→3 4→12→10→3 4→12→12→3 4→4→4→4 4→4→12→4
4→9→2→4 4→9→12→4 4→10→1→4 4→12→1→4 4→12→4→4 4→12→12→4
4→3→3→6 4→3→6→6 4→4→3→6 4→4→4→6 4→4→6→6 4→4→9→6
4→4→12→6 4→6→6→6 4→8→3→6 4→9→6→6 4→9→9→6 4→9→11→6
4→9→12→6 4→10→3→6 4→12→3→6 4→12→4→6 4→12→6→6 4→12→9→6
4→12→12→6 4→3→3→7 4→4→3→7 4→4→10→7 4→4→12→7 4→6→10→7
4→8→3→7 4→8→10→7 4→9→12→7 4→10→1→7 4→10→3→7 4→12→1→7
4→12→3→7 4→12→10→7 4→12→12→7 4→3→6→8 4→4→4→8 4→4→6→8
4→6→6→8 4→9→2→8 4→9→6→8 4→12→4→8 4→12→6→8 4→4→4→9
4→4→9→9 4→4→12→9 4→9→9→9 4→9→12→9 4→12→4→9 4→12→9→9
4→12→12→9 4→3→6→10 4→4→4→10 4→4→6→10 4→4→8→10 4→4→12→10
4→6→6→10 4→6→8→10 4→9→6→10 4→9→12→10 4→12→4→10 4→12→6→10
4→12→12→10 4→3→7→11 4→4→9→11 4→9→2→11 4→9→9→11 4→9→11→11
4→10→1→11 4→10→7→11 4→12→1→11 4→12→7→11 4→12→9→11
4→4→4→12 4→4→9→12 4→4→12→12 4→9→2→12 4→9→9→12 4→9→11→12
4→9→12→12 4→12→4→12 4→12→9→12 4→12→12→12 5→11→12→1
5→7→3→3 5→11→12→3 5→11→12→4 5→7→3→6 5→7→11→6 5→11→6→6
5→11→11→6 5→11→12→6 5→7→3→7 5→11→12→7 5→11→6→8 5→11→12→9
5→11→6→10 5→11→12→10 5→7→11→11 5→11→11→11 5→7→11→12
5→11→11→12 5→11→12→12 6→6→10→1 6→8→10→1 6→10→1→2 6→6→8→3
6→6→10→3 6→8→3→3 6→8→10→3 6→10→1→3 6→10→3→3 6→10→7→3
6→10→1→4 6→6→6→6 6→8→3→6 6→10→3→6 6→6→10→7 6→8→3→7
6→8→10→7 6→10→1→7 6→10→3→7 6→6→6→8 6→6→6→10 6→6→8→10
6→10→1→11 6→10→7→11 7→11→12→1 7→3→3→3 7→3→7→3 7→11→12→3
7→11→12→4 7→3→3→6 7→3→6→6 7→11→6→6 7→11→11→6 7→11→12→6
7→3→3→7 7→11→12→7 7→3→6→8 7→11→6→8 7→11→12→9 7→3→6→10
7→11→6→10 7→11→12→10 7→3→7→11 7→11→11→11 7→11→11→12
7→11→12→12 8→10→1→2 8→3→3→3 8→3→7→3 8→10→1→3 8→10→3→3
8→10→7→3 8→10→1→4 8→3→3→6 8→3→6→6 8→10→3→6 8→3→3→7
8→10→1→7 8→10→3→7 8→3→6→8 8→3→6→10 8→3→7→11 8→10→1→11
8→10→7→11 9→2→12→1 9→6→10→1 9→9→12→1 9→11→12→1 9→12→10→1
9→12→12→1 9→9→9→2 9→12→1→2 9→12→9→2 9→2→4→3 9→2→8→3

9→2→12→3 9→6→8→3 9→6→10→3 9→9→12→3 9→11→12→3 9→12→1→3
9→12→3→3 9→12→4→3 9→12→7→3 9→12→10→3 9→12→12→3 9→2→4→4
9→2→12→4 9→9→2→4 9→9→12→4 9→11→12→4 9→12→1→4 9→12→4→4
9→12→12→4 9→2→4→6 9→2→11→6 9→2→12→6 9→6→6→6 9→9→6→6
9→9→9→6 9→9→11→6 9→9→12→6 9→11→6→6 9→11→11→6 9→11→12→6
9→12→3→6 9→12→4→6 9→12→6→6 9→12→9→6 9→12→12→6 9→2→12→7
9→6→10→7 9→9→12→7 9→11→12→7 9→12→1→7 9→12→3→7 9→12→10→7
9→12→12→7 9→2→4→8 9→6→6→8 9→9→2→8 9→9→6→8 9→11→6→8
9→12→4→8 9→12→6→8 9→2→4→9 9→2→12→9 9→9→9→9 9→9→12→9
9→11→12→9 9→12→4→9 9→12→9→9 9→12→12→9 9→2→4→10 9→2→8→10
9→2→12→10 9→6→6→10 9→6→8→10 9→9→6→10 9→9→12→10 9→11→6→10
9→11→12→10 9→12→4→10 9→12→6→10 9→12→12→10 9→2→11→11
9→9→2→11 9→9→9→11 9→9→11→11 9→11→11→11 9→12→1→11
9→12→7→11 9→12→9→11 9→2→4→12 9→2→11→12 9→2→12→12 9→9→2→12
9→9→9→12 9→9→11→12 9→9→12→12 9→11→11→12 9→11→12→12
9→12→4→12 9→12→9→12 9→12→12→12 10→1→3→3 10→1→4→3 10→1→7→3
10→3→3→3 10→3→7→3 10→7→3→3 10→1→2→4 10→1→4→4 10→1→3→6
10→1→4→6 10→1→11→6 10→3→3→6 10→3→6→6 10→7→3→6 10→7→11→6
10→1→3→7 10→3→3→7 10→7→3→7 10→1→2→8 10→1→4→8 10→3→6→8
10→1→4→9 10→1→4→10 10→3→6→10 10→1→2→11 10→1→7→11
10→1→11→11 10→3→7→11 10→7→11→11 10→1→2→12 10→1→4→12
10→1→11→12 10→7→11→12 11→6→10→1 11→11→12→1 11→12→10→1
11→12→12→1 11→12→1→2 11→12→9→2 11→6→8→3 11→6→10→3
11→11→12→3 11→12→1→3 11→12→3→3 11→12→4→3 11→12→7→3
11→12→10→3 11→12→12→3 11→11→12→4 11→12→1→4 11→12→4→4
11→12→12→4 11→6→6→6 11→11→6→6 11→11→11→6 11→11→12→6
11→12→3→6 11→12→4→6 11→12→6→6 11→12→9→6 11→12→12→6
11→6→10→7 11→11→12→7 11→12→1→7 11→12→3→7 11→12→10→7
11→12→12→7 11→6→6→8 11→11→6→8 11→12→4→8 11→12→6→8
11→11→12→9 11→12→4→9 11→12→9→9 11→12→12→9 11→6→6→10
11→6→8→10 11→11→6→10 11→11→12→10 11→12→4→10 11→12→6→10
11→12→12→10 11→11→11→11 11→12→1→11 11→12→7→11 11→12→9→11
11→11→11→12 11→11→12→12 11→12→4→12 11→12→9→12 11→12→12→12
12→4→10→1 12→4→12→1 12→6→10→1 12→9→12→1 12→12→10→1
12→12→12→1 12→4→9→2 12→9→9→2 12→10→1→2 12→12→1→2
12→12→9→2 12→1→3→3 12→1→4→3 12→1→7→3 12→3→3→3 12→3→7→3
12→4→3→3 12→4→4→3 12→4→8→3 12→4→10→3 12→4→12→3 12→6→8→3
12→6→10→3 12→7→3→3 12→9→12→3 12→10→1→3 12→10→3→3
12→10→7→3 12→12→1→3 12→12→3→3 12→12→4→3 12→12→7→3
12→12→10→3 12→12→12→3 12→1→2→4 12→1→4→4 12→4→4→4
12→4→12→4 12→9→2→4 12→9→12→4 12→10→1→4 12→12→1→4
12→12→4→4 12→12→12→4 12→1→3→6 12→1→4→6 12→1→11→6 12→3→3→6
12→3→6→6 12→4→3→6 12→4→4→6 12→4→6→6 12→4→9→6 12→4→12→6
12→6→6→6 12→7→3→6 12→7→11→6 12→9→6→6 12→9→9→6 12→9→11→6
12→9→12→6 12→10→3→6 12→12→3→6 12→12→4→6 12→12→6→6
12→12→9→6 12→12→12→6 12→1→3→7 12→3→3→7 12→4→3→7 12→4→10→7
12→4→12→7 12→6→10→7 12→7→3→7 12→9→12→7 12→10→1→7
12→10→3→7 12→12→1→7 12→12→3→7 12→12→10→7 12→12→12→7
12→1→2→8 12→1→4→8 12→3→6→8 12→4→4→8 12→4→6→8 12→6→6→8
12→9→2→8 12→9→6→8 12→12→4→8 12→12→6→8 12→1→4→9 12→4→4→9
12→4→9→9 12→4→12→9 12→9→9→9 12→9→12→9 12→12→4→9 12→12→9→9
12→12→12→9 12→1→4→10 12→3→6→10 12→4→4→10 12→4→6→10
12→4→8→10 12→4→12→10 12→6→6→10 12→6→8→10 12→9→6→10
12→9→12→10 12→12→4→10 12→12→6→10 12→12→12→10 12→1→2→11
12→1→7→11 12→1→11→11 12→3→7→11 12→4→9→11 12→7→11→11
12→9→2→11 12→9→9→11 12→9→11→11 12→10→1→11 12→10→7→11
12→12→1→11 12→12→7→11 12→12→9→11 12→1→2→12 12→1→4→12
12→1→11→12 12→4→4→12 12→4→9→12 12→4→12→12 12→7→11→12
12→9→2→12 12→9→9→12 12→9→11→12 12→9→12→12 12→12→4→12

```
12→12→9→12 12→12→12→12
```

Матриця досяжності:

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

Матриця сильної зв'язаності

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

Компоненти сильної зв'язаності

```
[[1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12], [5]]
```

Конденсована матриця суміжності:

```
0 0
1 0
```

Зображення

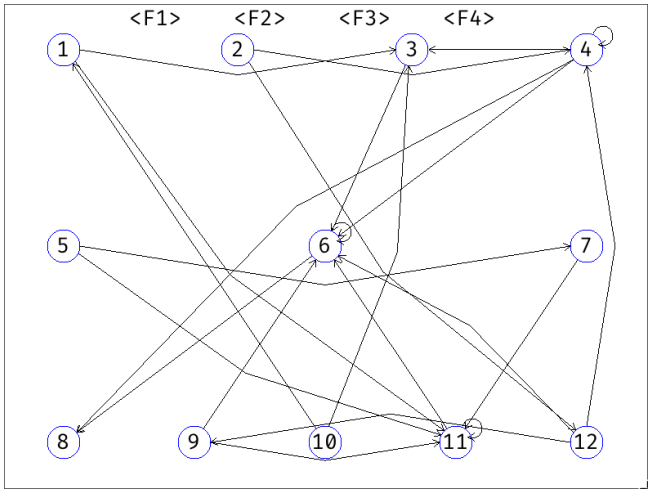

Рис. 1: Напрямлений граф

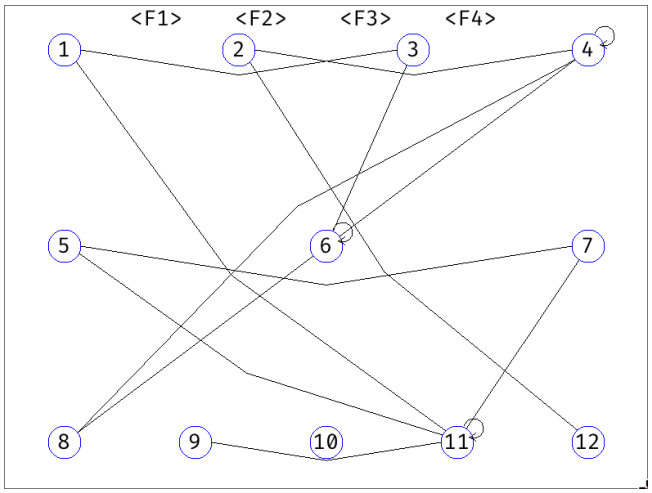

Рис. 2: Ненапрямлений граф

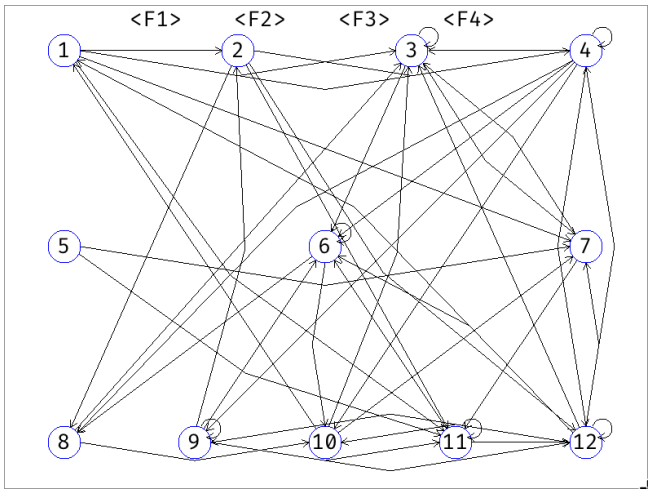

Рис. 3: Модифікований граф

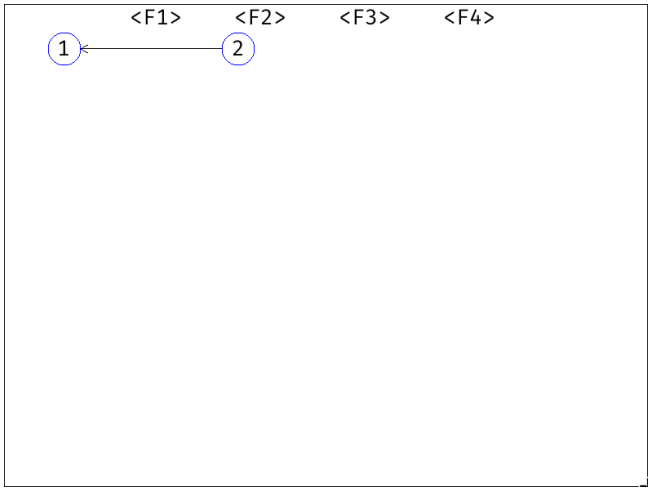

Рис. 4: Граф конденсації

Висновок

Модифікував програму з ЛР №3, щоб вона працювала з графами динамічного розміру. Застосував властивості графа для пошуку компонент сильної зв'язано-

сті та побудови графа конденсації.