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 №4

з дисципліни
«Алгоритми і структури даних»

Виконав:	Перевірив:
Студент групи ІМ-41	
Димура Ілля Олександрович	Сергієнко А. М.
Номер у списку групи: 7	

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:

    1) степені вершин напрямленого і ненапрямленого графів;

    2) напівстепені виходу та заходу напрямленого графа;

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

    4) перелік висячих та ізольованих вершин.

Результати вивести у графічне вікно, консоль або файл.

3. Змінити матрицю

    Коефіцієнт $k = 1.0 - n_3 * 0.005 - n_4 * 0.005 - 0.27$

4. Для нового орграфа обчислити:

    1) півстепені вершин;

    2) всі шляхи довжини 2 і 3;

    3) матрицю досяжності;

    4) матрицю сильної зв’язності;

    5) перелік компонент сильної зв’язності;

    6) граф конденсації.

    Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3 відповідно. Як результат вивести перелік шляхів, включно з усіма проміжними вершинами, через які проходить шлях.

Матрицю досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання. У переліку компонент слід вказати, які вершини належать до кожної компоненти. Граф конденсації вивести у графічне вікно.

Варіант 7:

$n_1n_2n_3n_4 = 4107$

Розміщення вершин: колом з вершиною в центрі

Кількість вершин: $10 + n_3 = 10$

Текст програми

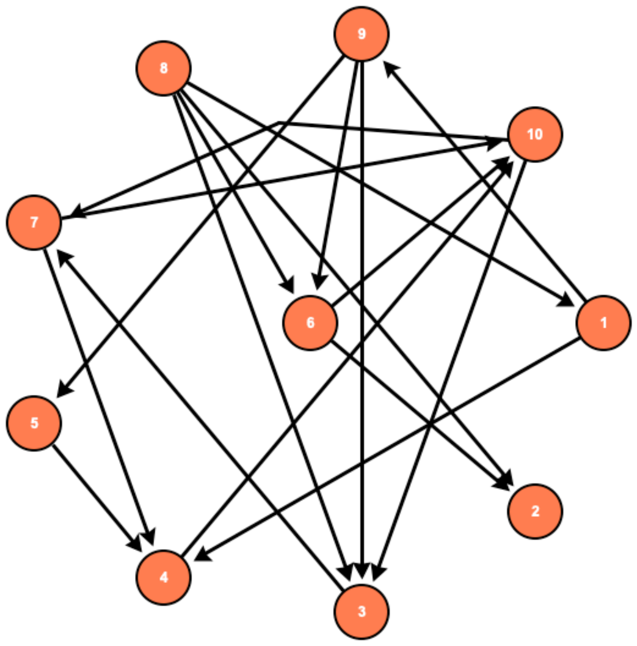
```python
import math

import random

import tkinter as tk

from typing import List, Tuple, Set, Sequence

from itertools import product


# types

Coord = Tuple[float, float]

Matrix = List[List[int]]


# config

VARIANT: int = 4107  # група ім-41, варіант 07

PANEL_SIZE: int = 600  # graph render size

PANEL_GAP: int = 40  # gap between graphs


OUTER_RADIUS: float = 0.40 * PANEL_SIZE

NODE_RADIUS: int = 22

EDGE_WIDTH: int = 3




def generate_directed_matrix(size: int,

                             seed: int,

                             n3: int,

                             n4: int) -> Matrix:

    random.seed(seed)

    k: float = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3

    return [

        [1 if random.uniform(0.0, 2.0) * k >= 1.0 else 0 for _ in range(size)]

        for _ in range(size)

    ]
```

```python
def to_undirected(matrix: Matrix) -> Matrix:
    n: int = len(matrix)
    result: Matrix = [row[:] for row in matrix]
    for i in range(n):
        for j in range(i + 1, n):
            result[i][j] = result[j][i] = 1 if matrix[i][j] or matrix[j][i] else 0
    return result


def print_matrix(matrix: Matrix, title: str) -> None:
    n: int = len(matrix)
    print(title)
    print("    " + "".join(f"{j + 1:>2}" for j in range(n)))
    print("    " + "-" * (3 * n))
    for i in range(n):
        row: str = " ".join(
            f"\033[31m{matrix[i][j]}\033[0m" if i == j else str(matrix[i][j])
            for j in range(n)
        )
        print(f"{i + 1:>2}| {row}")
    print()


def node_positions(count: int,
                   center_idx: int,
                   offset_x: int) -> List[Coord]:
    cx: float = offset_x + PANEL_SIZE / 2
    cy: float = PANEL_SIZE / 2
    positions: List[Coord] = [None] * count  # type: ignore
    positions[center_idx] = (cx, cy)
    outer: List[int] = [i for i in range(count) if i != center_idx]
    for k, idx in enumerate(outer):
        angle: float = 2 * math.pi * k / len(outer)
        positions[idx] = (cx + OUTER_RADIUS * math.cos(angle),
                          cy + OUTER_RADIUS * math.sin(angle))
    return positions


def shift_point(p: Coord, q: Coord, distance: float) -> Coord:
    dx: float = q[0] - p[0]
    dy: float = q[1] - p[1]
    length: float = math.hypot(dx, dy)
    return (p[0] + dx / length * distance,
            p[1] + dy / length * distance)


def draw_node(canvas: tk.Canvas, x: float, y: float, label: str) -> None:
    canvas.create_oval(x - NODE_RADIUS, y - NODE_RADIUS,
                       x + NODE_RADIUS, y + NODE_RADIUS,
                       fill="coral", outline="black", width=2)
    canvas.create_text(x, y, text=label, font=("Arial", 12, "bold"))


def draw_straight_edge(canvas: tk.Canvas,
                       p_from: Coord,
                       p_to: Coord,
                       with_arrow: bool) -> None:
    a: Coord = shift_point(p_from, p_to, NODE_RADIUS)
    b: Coord = shift_point(p_to, p_from, NODE_RADIUS * (1.25 if with_arrow else 1))
    canvas.create_line(*a, *b,
                       width=EDGE_WIDTH, fill="black",
                       arrow=tk.LAST if with_arrow else tk.NONE,
                       arrowshape=(12, 14, 6), capstyle=tk.ROUND)


def draw_angled_edge(canvas: tk.Canvas,
                     p_from: Coord,
                     p_to: Coord,
                     vertices: Sequence[Coord],
                     with_arrow: bool) -> None:
    dx, dy = p_to[0] - p_from[0], p_to[1] - p_from[1]
    length: float = math.hypot(dx, dy)
    a: Coord = (p_from[0] + dx / length * NODE_RADIUS,
                p_from[1] + dy / length * NODE_RADIUS)
    b: Coord = (p_to[0] - dx / length * NODE_RADIUS * 1.25,
                p_to[1] - dy / length * NODE_RADIUS * 1.25)

    mid: Coord = ((a[0] + b[0]) / 2, (a[1] + b[1]) / 2)
    offset: float = length * math.tan(0.035 * math.pi)

    best_midpoint: Coord | None = None
    best_clearance: float = -1.0
    for sign in (1, -1):
        perp: Coord = (-dy / length * offset * sign,
                       dx / length * offset * sign)
        m: Coord = (mid[0] + perp[0], mid[1] + perp[1])
        clearance: float = min(
            math.hypot(m[0] - vx, m[1] - vy)
            for (vx, vy) in vertices if (vx, vy) not in (p_from, p_to)
        )
        if clearance > best_clearance:
            best_clearance, best_midpoint = clearance, m  # type: ignore
    canvas.create_line(*a, *best_midpoint, width=EDGE_WIDTH,
                       fill="black", capstyle=tk.ROUND)
    canvas.create_line(*best_midpoint, *b, width=EDGE_WIDTH,
                       fill="black",
                       arrow=tk.LAST if with_arrow else tk.NONE,
                       arrowshape=(12, 14, 6), capstyle=tk.ROUND)


def draw_loop(canvas: tk.Canvas,
              x: float, y: float,
              with_arrow: bool) -> None:
    points: int = 16
    radius: float = NODE_RADIUS * 0.9
    start_a, end_a = -0.9 * math.pi, 0.55 * math.pi
    side: int = -1  # left
    sx, sy = x + side * NODE_RADIUS, y - 0.85 * NODE_RADIUS

    coords: List[float] = []
    for i in range(points):
        ang: float = start_a + (end_a - start_a) * i / (points - 1)
        coords += [sx + side * radius * math.cos(ang),
                   sy + radius * math.sin(ang)]

    canvas.create_line(*coords, smooth=True,
                       width=EDGE_WIDTH, fill="black",
                       arrow=tk.LAST if with_arrow else tk.NONE,
                       arrowshape=(12, 14, 6), capstyle=tk.ROUND)


def render_matrix(c: tk.Canvas,
                  mx: Matrix,
                  origin: Coord,
                  ):
    ox, oy = origin
    n = len(mx)
    line_h = 20
    c.create_text(ox, oy, anchor="w", fill="black", text="Adjacency matrix:",
font=("Arial", 19, "bold"))
    oy += line_h + 5
    for i in range(n):
        row_str = " ".join(
            f"{mx[i][j]}" for j in range(n)
        )
        c.create_text(ox, oy + i * line_h, anchor="w",
                      text=row_str, fill="black", font=("Arial", 18, "bold"))


def degrees_undirected(mx: Matrix) -> List[int]:
    return [sum(row) for row in mx]


def degrees_directed(mx: Matrix) -> Tuple[List[int], List[int], List[int]]:
    out_deg = [sum(row) for row in mx]
    in_deg = [sum(col) for col in zip(*mx)]
    total = [o + i for o, i in zip(out_deg, in_deg)]
    return out_deg, in_deg, total


def regular_degree(degs: List[int]) -> int | None:
    return degs[0] if all(d == degs[0] for d in degs) else None


def pendant_isolated(degs: List[int]) -> Tuple[List[int], List[int]]:
    pend = [i + 1 for i, d in enumerate(degs) if d == 1]
    isol = [i + 1 for i, d in enumerate(degs) if d == 0]
    return pend, isol


def generate_directed_matrix_v2(size: int,
                                seed: int,
                                n3: int,
                                n4: int) -> Matrix:
    random.seed(seed)
    k = 1.0 - n3 * 0.005 - n4 * 0.005 - 0.27
    return [
        [1 if random.uniform(0.0, 2.0) * k >= 1.0 else 0 for _ in range(size)]
        for _ in range(size)
    ]


def multiply_matrices(a: Matrix, b: Matrix) -> Matrix:
    n = len(a)
    result = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            result[i][j] = sum(a[i][k] * b[k][j] for k in range(n))
    return result


def enumerate_paths_len2(mx: Matrix) -> List[Tuple[int, int, int]]:
    n = len(mx)
    a2 = multiply_matrices(mx, mx)
    paths = []

    for i in range(n):
        for j in range(n):
            if a2[i][j]:
                for k in range(n):
                    if mx[i][k] and mx[k][j]:
                        paths.append((i, k, j))
    return paths


def enumerate_paths_len3(mx: Matrix) -> List[Tuple[int, int, int, int]]:
    n = len(mx)
    a2 = multiply_matrices(mx, mx)
    a3 = multiply_matrices(a2, mx)
    paths = []

    for i in range(n):
        for j in range(n):
            if a3[i][j]:
                for k in range(n):
                    if mx[i][k]:
                        for l in range(n):
                            if mx[k][l] and mx[l][j]:
                                paths.append((i, k, l, j))
    return paths


def reachability_matrix(mx: Matrix) -> Matrix:
    n = len(mx)
    r = [row[:] for row in mx]
    for i in range(n):
        r[i][i] = 1
    for k in range(n):
        for i in range(n):
            if r[i][k]:
                for j in range(n):
                    if r[k][j]:
                        r[i][j] = 1
    return r


def strongly_connected_components(r: Matrix) -> Tuple[List[List[int]], List[int]]:
    n = len(r)
    comp_id = [-1] * n
    comps: List[List[int]] = []
    cid = 0
    for v in range(n):
        if comp_id[v] != -1:
            continue
        comp = [u for u in range(n) if r[v][u] and r[u][v]]
        for u in comp:
            comp_id[u] = cid
        comps.append(comp)
        cid += 1
    return comps, comp_id


def condensation_matrix(mx: Matrix, comp_id: List[int], comp_count: int) -> Matrix:
    res = [[0] * comp_count for _ in range(comp_count)]
    n = len(mx)
    for i, j in product(range(n), repeat=2):
        if mx[i][j]:
            ci, cj = comp_id[i], comp_id[j]
            if ci != cj:
                res[ci][cj] = 1
    return res
```

```python
def draw_graph(canvas: tk.Canvas,
               positions: List[Coord],
               matrix: Matrix,
               offset_x: int,
               directed: bool,
               caption: str | None = None) -> None:
    processed: Set[Tuple[int, int]] = set()
    count: int = len(matrix)

    for i in range(count):
        traversal = range(count) if directed else range(i, count)
        for j in traversal:
            if not matrix[i][j]:
                continue
            if i == j:
                draw_loop(canvas, *positions[i], with_arrow=True)
                continue

            if directed and matrix[j][i] and (j, i) not in processed:
                draw_straight_edge(canvas, positions[i], positions[j], True)
                draw_angled_edge(canvas, positions[j], positions[i],
                                 positions, True)
                processed.update({(i, j), (j, i)})
            elif not directed or (i, j) not in processed:
                draw_straight_edge(canvas, positions[i], positions[j], directed)
                if directed:
                    processed.add((i, j))

    for idx, (x, y) in enumerate(positions):
        draw_node(canvas, x, y, str(idx + 1))

    cap = caption or ("Directed" if directed else "Undirected")
    canvas.create_text(offset_x + PANEL_SIZE / 2, PANEL_SIZE - 15,
                       text=cap, fill="black", font=("Arial", 17, "bold"))

    render_matrix(canvas, matrix,
                  origin=(offset_x + PANEL_SIZE / 4, PANEL_SIZE + 10),
                  )


def print_graph_info(mx: Matrix, directed: bool, label: str):
    print()
    print_matrix(mx, f"{label}: Adjacency matrix")
    if directed:
        out_d, in_d, tot_d = degrees_directed(mx)
        print("Out semi-degrees :", out_d)
        print("In semi-degrees  :", in_d)
    else:
        tot_d = degrees_undirected(undirected_mx)
    print("Degrees:", tot_d)
    pend_d, iso_d = pendant_isolated(tot_d)
    print(f"Pendant vertices: {pend_d or 'none'}")
    print(f"Isolated vertices: {iso_d or 'none'}")
    reg = regular_degree(tot_d)
    print(f"Graph is", f"regular of degree {reg}" if reg is not None else "NOT
regular")
    print()


if __name__ == "__main__":
    n1, n2, n3, n4 = map(int, str(VARIANT).zfill(4))
    vertex_count: int = 10 + n3

    directed_mx: Matrix = generate_directed_matrix(vertex_count, VARIANT, n3, n4)
    undirected_mx: Matrix = to_undirected(directed_mx)
    print_graph_info(directed_mx, True, 'Directed graph')
    print_graph_info(undirected_mx, False, 'Undirected graph')

    directed_mx2 = generate_directed_matrix_v2(vertex_count, VARIANT, n3, n4)
    print_graph_info(directed_mx2, True, 'New directed graph')

    a2 = multiply_matrices(directed_mx2, directed_mx2)
    a3 = multiply_matrices(a2, directed_mx2)

    paths2 = enumerate_paths_len2(directed_mx2)
    paths3 = enumerate_paths_len3(directed_mx2)
    print("Paths length 2: ", ", ".join(f"{i + 1} → {k + 1} → {j + 1}" for i, k, j
in paths2))
    print("\nPaths length 3: ", ", ".join(f"{i + 1} → {k + 1} → {l + 1} → {j + 1}"
for i, k, l, j in paths3))

    reach = reachability_matrix(directed_mx2)
    print_matrix(reach, "Reachability matrix")

    scc_mx = [[1 if reach[i][j] and reach[j][i] else 0
               for j in range(vertex_count)] for i in range(vertex_count)]
    print_matrix(scc_mx, "Strong-connectivity matrix")

    comps, comp_of = strongly_connected_components(reach)
    print("Strongly-connected components:")
    for idx, comp in enumerate(comps, 1):
        members = ", ".join(str(v + 1) for v in comp)
        print(f"  C{idx}: {{ {members} }}")

    cond_mx = condensation_matrix(directed_mx2, comp_of, len(comps))
    print_matrix(cond_mx, "Condensed graph: Adjacency matrix")
```

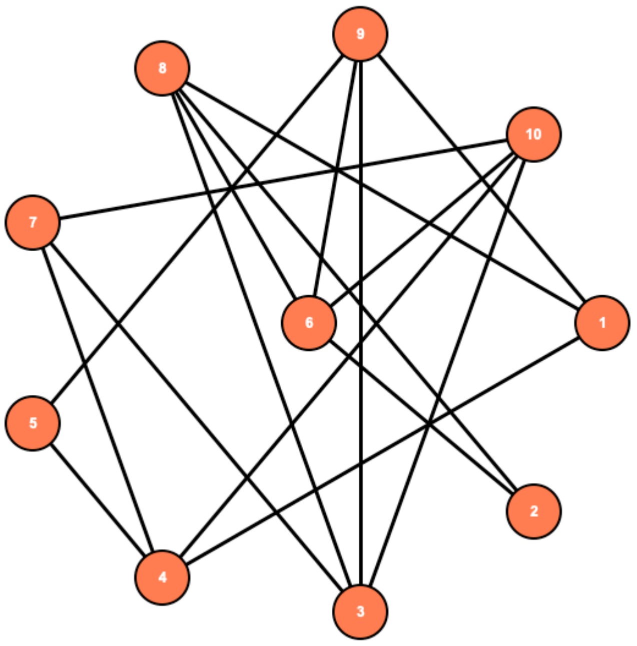
```python
# draw

canvas_width = 4 * PANEL_SIZE + 2 * PANEL_GAP  # 3 панелі
root = tk.Tk()
root.title(f"Lab 4 · Variant {VARIANT}")
canvas = tk.Canvas(root,
                   width=canvas_width,
                   height=PANEL_SIZE + 330,
                   bg="white")
canvas.pack()

pos_dir = node_positions(vertex_count, vertex_count // 2, offset_x=0)
pos_undir = node_positions(vertex_count, vertex_count // 2,
                           offset_x=PANEL_SIZE + PANEL_GAP)
pos_new_dir = node_positions(vertex_count, vertex_count // 2,
                             offset_x=2 * PANEL_SIZE + PANEL_GAP)
pos_cond = node_positions(len(comps), len(comps) // 2,
                          offset_x=3 * PANEL_SIZE + 2 * PANEL_GAP)

draw_graph(canvas, pos_dir, directed_mx, offset_x=0,
           directed=True, caption="Original directed")
draw_graph(canvas, pos_undir, undirected_mx,
           offset_x=PANEL_SIZE + PANEL_GAP,
           directed=False, caption="Undirected")
draw_graph(canvas, pos_new_dir, directed_mx2,
           offset_x=2 * PANEL_SIZE + 2 * PANEL_GAP,
           directed=True, caption="New directed")
draw_graph(canvas, pos_cond, cond_mx,
           offset_x=3 * PANEL_SIZE + 2 * PANEL_GAP,
           directed=True, caption="Condensed graph")

root.mainloop()
```

Тести програми:



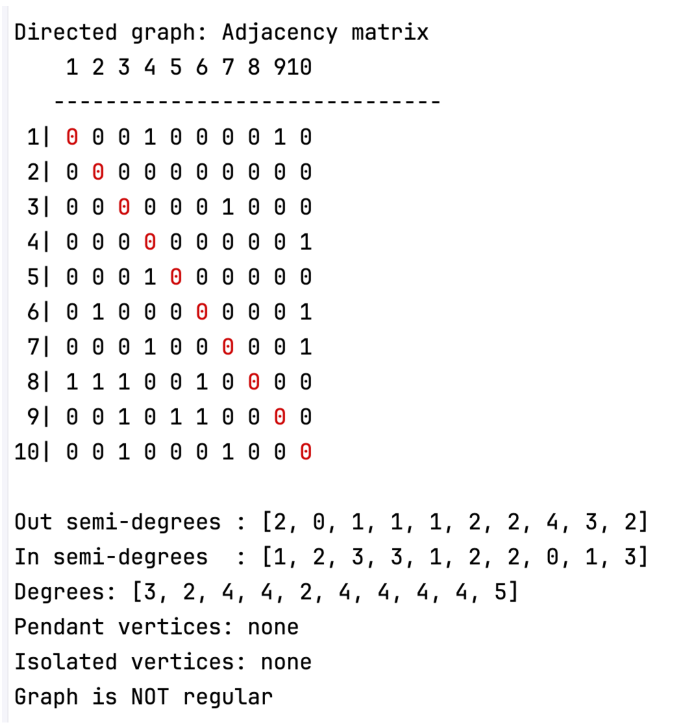
Original directed

```
Adjacency matrix:
0 0 0 1 0 0 0 0 1 0
0 0 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 1
0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 1
1 1 1 0 0 1 0 0 0 0
0 0 1 0 1 1 0 0 0 0
0 0 1 0 0 0 1 0 0 0
```

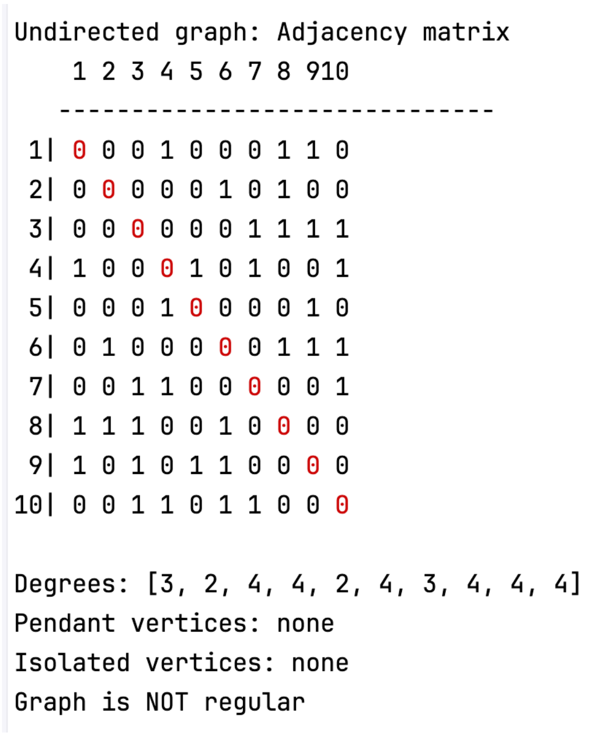


Undirected

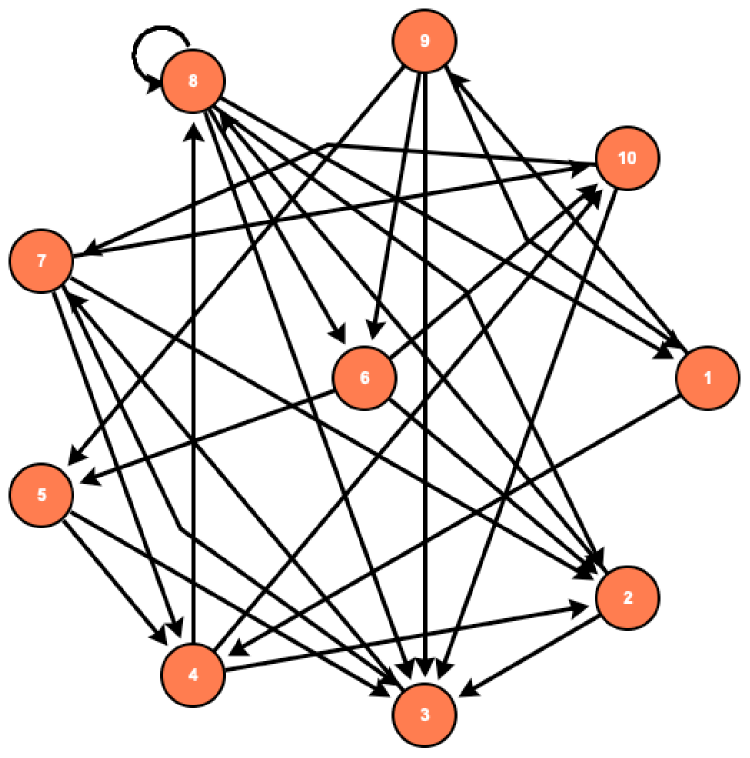
```
Adjacency matrix:
0 0 0 1 0 0 0 1 1 0
0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 1 1 1 1
1 0 0 0 1 0 1 0 0 1
0 0 0 1 0 0 0 0 1 0
0 1 0 0 0 0 0 1 1 1
0 0 1 1 0 0 0 0 0 1
1 1 1 0 0 1 0 0 0 0
1 0 1 0 1 1 0 0 0 0
0 0 1 1 0 1 1 0 0 0
```



```
Directed graph: Adjacency matrix
    1 2 3 4 5 6 7 8 910
   ----------------------------
 1| 0 0 0 1 0 0 0 0 1 0
 2| 0 0 0 0 0 0 0 0 0 0
 3| 0 0 0 0 0 0 1 0 0 0
 4| 0 0 0 0 0 0 0 0 0 1
 5| 0 0 0 1 0 0 0 0 0 0
 6| 0 1 0 0 0 0 0 0 0 1
 7| 0 0 0 1 0 0 0 0 0 1
 8| 1 1 1 0 0 1 0 0 0 0
 9| 0 0 1 0 1 1 0 0 0 0
10| 0 0 1 0 0 0 1 0 0 0

Out semi-degrees : [2, 0, 1, 1, 1, 2, 2, 4, 3, 2]
In semi-degrees  : [1, 2, 3, 3, 1, 2, 2, 0, 1, 3]
Degrees: [3, 2, 4, 4, 2, 4, 4, 4, 4, 5]
Pendant vertices: none
Isolated vertices: none
Graph is NOT regular
```



```
Undirected graph: Adjacency matrix
    1 2 3 4 5 6 7 8 910
   ----------------------------
 1| 0 0 0 1 0 0 0 1 1 0
 2| 0 0 0 0 0 1 0 1 0 0
 3| 0 0 0 0 0 0 1 1 1 1
 4| 1 0 0 0 1 0 1 0 0 1
 5| 0 0 0 1 0 0 0 0 1 0
 6| 0 1 0 0 0 0 0 1 1 1
 7| 0 0 1 1 0 0 0 0 0 1
 8| 1 1 1 0 0 1 0 0 0 0
 9| 1 0 1 0 1 1 0 0 0 0
10| 0 0 1 1 0 1 1 0 0 0

Degrees: [3, 2, 4, 4, 2, 4, 3, 4, 4, 4]
Pendant vertices: none
Isolated vertices: none
Graph is NOT regular
```



New directed

Adjacency matrix:

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

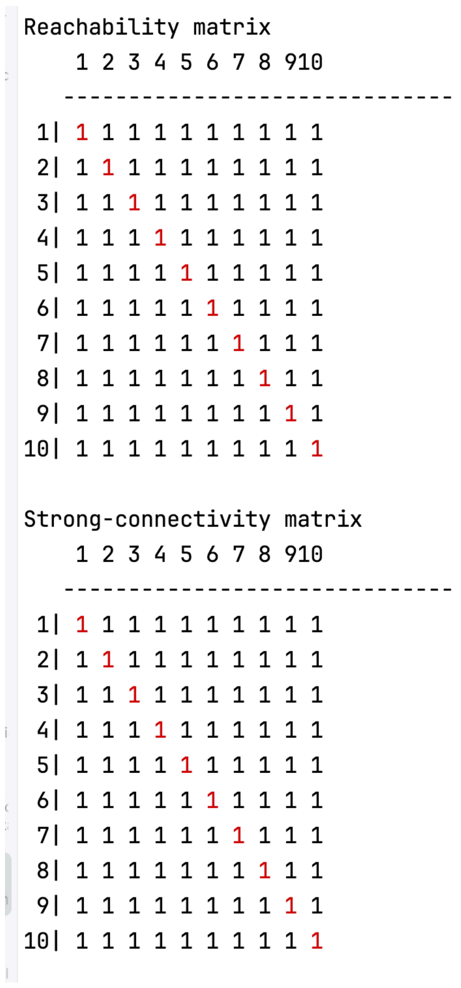
Condensed graph

Adjacency matrix:

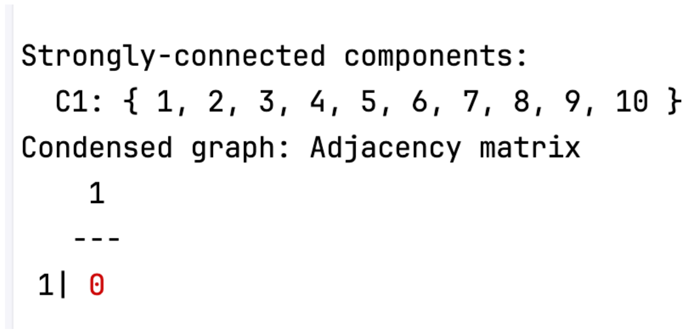
0

```
New directed graph: Adjacency matrix
    1 2 3 4 5 6 7 8 910
   -----------------------------
 1| 0 0 0 1 0 0 0 0 1 0
 2| 0 0 1 0 0 0 0 1 0 0
 3| 0 0 0 0 0 0 1 0 0 0
 4| 0 1 0 0 0 0 0 1 0 1
 5| 0 0 1 1 0 0 0 0 0 0
 6| 0 1 0 0 1 0 0 0 0 1
 7| 0 1 1 1 0 0 0 0 0 1
 8| 1 1 1 0 0 1 0 1 0 0
 9| 1 0 1 0 1 1 0 0 0 0
10| 0 0 1 0 0 0 1 0 0 0

Out semi-degrees : [2, 2, 1, 3, 2, 3, 4, 5, 4, 2]
In semi-degrees  : [2, 4, 6, 3, 2, 2, 2, 3, 1, 3]
Degrees: [4, 6, 7, 6, 4, 5, 6, 8, 5, 5]
Pendant vertices: none
Isolated vertices: none
Graph is NOT regular
```

```
Paths length 2:  1 → 4 → 2, 1 → 4 → 8, 1 → 4 → 10, 1 → 9 → 1, 1 → 9 → 3, 1 → 9 → 5, 1 → 9 → 6, 2 → 3 → 7, 2 → 8 → 1, 2 → 8 → 2, 2 → 8 → 3, 2 → 8 → 6, 2 → 8 → 8, 3
→ 7 → 2, 3 → 7 → 3, 3 → 7 → 4, 3 → 7 → 10, 4 → 2 → 3, 4 → 2 → 8, 4 → 8 → 1, 4 → 8 → 2, 4 → 8 → 3, 4 → 8 → 6, 4 → 8 → 8, 4 → 10 → 3, 4 → 10 → 7, 5 → 3 → 7, 5 → 4 →
2, 5 → 4 → 8, 5 → 4 → 10, 6 → 2 → 3, 6 → 2 → 8, 6 → 5 → 3, 6 → 5 → 4, 6 → 10 → 3, 6 → 10 → 7, 7 → 2 → 3, 7 → 2 → 8, 7 → 3 → 7, 7 → 4 → 2, 7 → 4 → 8, 7 → 4 → 10, 7
→ 10 → 3, 7 → 10 → 7, 8 → 1 → 4, 8 → 1 → 9, 8 → 2 → 3, 8 → 2 → 8, 8 → 3 → 7, 8 → 6 → 2, 8 → 6 → 5, 8 → 6 → 10, 8 → 8 → 1, 8 → 8 → 2, 8 → 8 → 3, 8 → 8 → 6, 8 → 8 →
8, 9 → 1 → 4, 9 → 1 → 9, 9 → 3 → 7, 9 → 5 → 3, 9 → 5 → 4, 9 → 6 → 2, 9 → 6 → 5, 9 → 6 → 10, 10 → 3 → 7, 10 → 7 → 2, 10 → 7 → 3, 10 → 7 → 4, 10 → 7 → 10

Paths length 3:  1 → 4 → 2 → 3, 1 → 4 → 2 → 8, 1 → 4 → 8 → 1, 1 → 4 → 8 → 2, 1 → 4 → 8 → 3, 1 → 4 → 8 → 6, 1 → 4 → 8 → 8, 1 → 4 → 10 → 3, 1 → 4 → 10 → 7, 1 → 9 → 1
 → 4, 1 → 9 → 1 → 9, 1 → 9 → 3 → 7, 1 → 9 → 5 → 3, 1 → 9 → 5 → 4, 1 → 9 → 6 → 2, 1 → 9 → 6 → 5, 1 → 9 → 6 → 10, 2 → 3 → 7 → 2, 2 → 3 → 7 → 3, 2 → 3 → 7 → 4, 2 → 3
→ 7 → 10, 2 → 8 → 1 → 4, 2 → 8 → 1 → 9, 2 → 8 → 2 → 3, 2 → 8 → 2 → 8, 2 → 8 → 3 → 7, 2 → 8 → 6 → 2, 2 → 8 → 6 → 5, 2 → 8 → 6 → 10, 2 → 8 → 8 → 1, 2 → 8 → 8 → 2, 2
→ 8 → 8 → 3, 2 → 8 → 8 → 6, 2 → 8 → 8 → 8, 3 → 7 → 2 → 3, 3 → 7 → 2 → 8, 3 → 7 → 3 → 7, 3 → 7 → 4 → 2, 3 → 7 → 4 → 8, 3 → 7 → 4 → 10, 3 → 7 → 10 → 3, 3 → 7 → 10 →
7, 4 → 2 → 3 → 7, 4 → 2 → 8 → 1, 4 → 2 → 8 → 2, 4 → 2 → 8 → 3, 4 → 2 → 8 → 6, 4 → 2 → 8 → 8, 4 → 8 → 1 → 4, 4 → 8 → 1 → 9, 4 → 8 → 2 → 3, 4 → 8 → 2 → 8, 4 → 8 → 3
→ 7, 4 → 8 → 6 → 2, 4 → 8 → 6 → 5, 4 → 8 → 6 → 10, 4 → 8 → 8 → 1, 4 → 8 → 8 → 2, 4 → 8 → 8 → 3, 4 → 8 → 8 → 6, 4 → 8 → 8 → 8, 4 → 10 → 3 → 7, 4 → 10 → 7 → 2, 4 → 1
0 → 7 → 3, 4 → 10 → 7 → 4, 4 → 10 → 7 → 10, 5 → 3 → 7 → 2, 5 → 3 → 7 → 3, 5 → 3 → 7 → 4, 5 → 3 → 7 → 10, 5 → 4 → 2 → 3, 5 → 4 → 2 → 8, 5 → 4 → 8 → 1, 5 → 4 → 8 → 2
, 5 → 4 → 8 → 3, 5 → 4 → 8 → 6, 5 → 4 → 8 → 8, 5 → 4 → 10 → 3, 5 → 4 → 10 → 7, 6 → 2 → 3 → 7, 6 → 2 → 8 → 1, 6 → 2 → 8 → 2, 6 → 2 → 8 → 3, 6 → 2 → 8 → 6, 6 → 2 → 8
 → 8, 6 → 5 → 3 → 7, 6 → 5 → 4 → 2, 6 → 5 → 4 → 8, 6 → 5 → 4 → 10, 6 → 10 → 3 → 7, 6 → 10 → 7 → 2, 6 → 10 → 7 → 3, 6 → 10 → 7 → 4, 6 → 10 → 7 → 10, 7 → 2 → 3 → 7,
7 → 2 → 8 → 1, 7 → 2 → 8 → 2, 7 → 2 → 8 → 3, 7 → 2 → 8 → 6, 7 → 2 → 8 → 8, 7 → 3 → 7 → 2, 7 → 3 → 7 → 3, 7 → 3 → 7 → 4, 7 → 3 → 7 → 10, 7 → 4 → 2 → 3, 7 → 4 → 2 →
8, 7 → 4 → 8 → 1, 7 → 4 → 8 → 2, 7 → 4 → 8 → 3, 7 → 4 → 8 → 6, 7 → 4 → 8 → 8, 7 → 4 → 10 → 3, 7 → 4 → 10 → 7, 7 → 10 → 3 → 7, 7 → 10 → 7 → 2, 7 → 10 → 7 → 3, 7 → 1
0 → 7 → 4, 7 → 10 → 7 → 10, 8 → 1 → 4 → 2, 8 → 1 → 4 → 8, 8 → 1 → 4 → 10, 8 → 1 → 9 → 1, 8 → 1 → 9 → 3, 8 → 1 → 9 → 5, 8 → 1 → 9 → 6, 8 → 2 → 3 → 7, 8 → 2 → 8 → 1,
 8 → 2 → 8 → 2, 8 → 2 → 8 → 3, 8 → 2 → 8 → 6, 8 → 2 → 8 → 8, 8 → 3 → 7 → 2, 8 → 3 → 7 → 3, 8 → 3 → 7 → 4, 8 → 3 → 7 → 10, 8 → 6 → 2 → 3, 8 → 6 → 2 → 8, 8 → 6 → 5 →
 3, 8 → 6 → 5 → 4, 8 → 6 → 10 → 3, 8 → 6 → 10 → 7, 8 → 8 → 1 → 4, 8 → 8 → 1 → 9, 8 → 8 → 2 → 3, 8 → 8 → 2 → 8, 8 → 8 → 3 → 7, 8 → 8 → 6 → 2, 8 → 8 → 6 → 5, 8 → 8 →
 6 → 10, 8 → 8 → 8 → 1, 8 → 8 → 8 → 2, 8 → 8 → 8 → 3, 8 → 8 → 8 → 6, 8 → 8 → 8 → 8, 9 → 1 → 4 → 2, 9 → 1 → 4 → 8, 9 → 1 → 4 → 10, 9 → 1 → 9 → 1, 9 → 1 → 9 → 3, 9 →
 1 → 9 → 5, 9 → 1 → 9 → 6, 9 → 3 → 7 → 2, 9 → 3 → 7 → 3, 9 → 3 → 7 → 4, 9 → 3 → 7 → 10, 9 → 5 → 3 → 7, 9 → 5 → 4 → 2, 9 → 5 → 4 → 8, 9 → 5 → 4 → 10, 9 → 6 → 2 → 3,
 9 → 6 → 2 → 8, 9 → 6 → 5 → 3, 9 → 6 → 5 → 4, 9 → 6 → 10 → 3, 9 → 6 → 10 → 7, 10 → 3 → 7 → 2, 10 → 3 → 7 → 3, 10 → 3 → 7 → 4, 10 → 3 → 7 → 10, 10 → 7 → 2 → 3, 10 →
 7 → 2 → 8, 10 → 7 → 3 → 7, 10 → 7 → 4 → 2, 10 → 7 → 4 → 8, 10 → 7 → 4 → 10, 10 → 7 → 10 → 3, 10 → 7 → 10 → 7
```



```
Reachability matrix
    1 2 3 4 5 6 7 8 910
   ------------------------------
 1| 1 1 1 1 1 1 1 1 1 1
 2| 1 1 1 1 1 1 1 1 1 1
 3| 1 1 1 1 1 1 1 1 1 1
 4| 1 1 1 1 1 1 1 1 1 1
 5| 1 1 1 1 1 1 1 1 1 1
 6| 1 1 1 1 1 1 1 1 1 1
 7| 1 1 1 1 1 1 1 1 1 1
 8| 1 1 1 1 1 1 1 1 1 1
 9| 1 1 1 1 1 1 1 1 1 1
10| 1 1 1 1 1 1 1 1 1 1

Strong-connectivity matrix
    1 2 3 4 5 6 7 8 910
   ------------------------------
 1| 1 1 1 1 1 1 1 1 1 1
 2| 1 1 1 1 1 1 1 1 1 1
 3| 1 1 1 1 1 1 1 1 1 1
 4| 1 1 1 1 1 1 1 1 1 1
 5| 1 1 1 1 1 1 1 1 1 1
 6| 1 1 1 1 1 1 1 1 1 1
 7| 1 1 1 1 1 1 1 1 1 1
 8| 1 1 1 1 1 1 1 1 1 1
 9| 1 1 1 1 1 1 1 1 1 1
10| 1 1 1 1 1 1 1 1 1 1
```



```
Strongly-connected components:
  C1: { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
Condensed graph: Adjacency matrix
    1
   ---
 1| 0
```

Висновок:

Лабораторна робота була виконана на мові програмування Python, використовуючи кросплатформну бібліотеку tkinter. Додав нові функції для обчислень та модифікував код рендеру, щоб вивести в вікно нові графи