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:
студент групи ІМ-31
Литвиненко Сергій Андрійович
номер у списку групи: 12

Перевірила:
Молчанова А. А.

Київ 2024

Постановка задачі

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами так само, як у лабораторній роботі №3.

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

Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$;

2) матриця розміром n * n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$, кожен елемент матриці множиться на коефіцієнт k;

4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

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 A_{dir} , 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.

Результати вивести у графічне вікно, в консоль або файл.

Шляхи довжиною 2 і 3 слід шукати за матрицями A2 і A3 , 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 12

n1 = 3, n2 = 1, n3 = 1, n4 = 2;

Кількість вершин - 10 + n3 = 11;

Розміщення вершин – квадрат (прямокутник);

Текст програми

Файл matrix/graph.py

```
import math
import random
from config import seed
from utils import float_random, index_filter
from .operations import sum_row, sum_col

def adjacency_matrix(size, k):
  random.seed(seed)
  matrix = []
  for _ in range(size):
    row = []
    for _ in range(size):
      val = math.floor(float_random(0, 2) * k)
      row.append(val)
    matrix.append(row)
  return matrix

def to_indirected(matrix):
  length = len(matrix)
  result = []
  for i in range(length):
    result.append([0] * length)
    for j in range(i + 1):
      value = matrix[i][j] | matrix[j][i]
      result[i][j] = value
      result[j][i] = value
  return result

def vertex_degree(matrix, i, directed):
  out = sum_row(matrix[i])
  incoming = sum_col(matrix, i) if directed else matrix[i][i]
  return out + incoming
```

```
def vertex_degrees(matrix, directed):
  result = []
  for i in range(len(matrix)):
    result.append(vertex_degree(matrix, i, directed))
  return result

vertex_degree_in = lambda matrix, i: sum_col(matrix, i)

vertex_degree_out = lambda matrix, i: sum_row(matrix[i])

vertex_degrees_in = lambda matrix: [
  sum_col(matrix, i) for i in range(len(matrix))
]

vertex_degrees_out = lambda matrix: [sum_row(row) for row in matrix]

def is_regular(degrees):
  degree = degrees[0]
  for i in range(1, len(degrees)):
    if degrees[i] != degree:
      return False
  return True

hanging_vertices = lambda degrees: index_filter(degrees, lambda x: x == 1)

isolated_vertices = lambda degrees: index_filter(degrees, lambda x: x ==
0)

def exist_edge(path, v1, v2):
  for i in range(len(path) - 1):
    if path[i] == v1 and path[i + 1] == v2:
      return True
  return False
```

```
def wrap_route(matrix, start, end, length, possible, path, result):
  path.append(start)
  if length <= 1:
    if matrix[start][end] and not exist_edge(path, start, end):
      result.append((*path, end))
    return result
  for i in range(len(matrix)):
    if matrix[start][i] == 0 or exist_edge(path, start, i):
      continue
    if len(result) >= possible:
      return result
    wrap_route(matrix, i, end, length - 1, possible, path, result)
    path.pop()
  return result

def route(matrix, start, end, length, possible):
  return wrap_route(matrix, start, end, length, possible, [], [])

def routes(adjancy, matrix, length):
  result = []
  items = len(matrix)
  for i in range(items):
    for j in range(items):
      if matrix[i][j] == 0: continue
      paths = route(adjancy, i, j, length, matrix[i][j])
      result.extend(paths)
  return result
```

Файл matrix/operations.py

```
def mul_matrix(m1, m2):
  rows = len(m1)
  cols = len(m2[0])
  result = []
  for i in range(rows):
    row = []
    for j in range(cols):
      sum = 0
      for k in range(rows):
        sum += m1[i][k] * m2[k][j]
      row.append(sum)
    result.append(row)
  return result

def sum_col(matrix, i):
  result = 0
  for j in range(len(matrix)):
    result += matrix[j][i]
  return result

def sum_row(row):
  result = 0
  for n in row:
    result += n
  return result

def union(matrix1, matrix2):
  result = []
  length = len(matrix1)
  for i in range(length):
    row = []
    for j in range(length):
      row.append(matrix1[i][j] or matrix2[i][j])
```

```
    result.append(row)
  return result

def compose(matrix1, matrix2):
  result = []
  length = len(matrix1)
  for i in range(length):
    row = [0] * length
    for j in range(length):
      for k in range(length):
        if matrix1[i][k] and matrix2[k][j]:
          row[j] = 1
          break
    result.append(row)
  return result

def transitive_closure(matrix, n):
  result = matrix
  p_matrix = matrix
  for _ in range(n - 1):
    p_matrix = compose(p_matrix, matrix)
    result = union(result, p_matrix)
  return result

def strong_connectivity(reachability):
  result = []
  length = len(reachability)
  for i in range(length):
    row = []
    for j in range(length):
      row.append(reachability[i][j] and reachability[j][i])
    result.append(row)
  return result
```

```
def strong_connectivity_items(strong_matrix):
  result = []
  all_zeros = hash((0,) * len(strong_matrix))
  table = {}
  for i, row in enumerate(strong_matrix):
    h = hash(tuple(row))
    if h == all_zeros:
      result.append([i])
    index = table.get(h, -1)
    if index == -1:
      result.append([i])
      table[h] = i
    else:
      result[index].append(i)
  return result

def chained_strong_items(matrix, strong_items):
  result = []
  for i, first in enumerate(strong_items):
    for j, second in enumerate(strong_items):
      if i == j: continue
      flag = False
      for m in first:
        for n in second:
          if matrix[m][n]:
            result.append((i, j))
            flag = True
            break
        if flag: break
  return result

def condensation_matrix(matrix, strong_items):
  items = chained_strong_items(matrix, strong_items)
  length = len(strong_items)
```

```
result = []
for i in range(length):
  row = []
  for j in range(length):
    data = 1 if (i, j) in items else 0
    row.append(data)
  result.append(row)
return result
```

Файл matrix/print.py

```
import csv
from utils import partial, in_one_line, is_neighbors, minmax
from vertex import (
  get_vertex_closure, vertex_draw, vertex_loop, vertex_arc, vertex_line
)

def draw_graph(canvas, matrix, sides, directed=True):
  length = len(matrix)
  one_line = partial(in_one_line, length, sides)
  neighbors = partial(is_neighbors, length)
  get_vertex = get_vertex_closure(length, sides)
  for i in range(length):
    vertex = get_vertex(i)
    row = matrix[i]
    vertex_draw(canvas, vertex)
    count = length if directed else i + 1
    for j in range(count):
      if row[j] != 1: continue
      other_vertex = get_vertex(j)
      m, n = minmax(i, j)
      if m == n:
        vertex_loop(canvas, vertex, directed)
      elif not neighbors(m, n) and one_line(m, n):
        vertex_arc(canvas, vertex, other_vertex, directed)
      else:
        shift = j < i and matrix[j][i] and directed
        vertex_line(canvas, vertex, other_vertex, shift, directed)

def print_matrix(matrix):
  width = 5
  intend = 3
  print(width * ' ', end='')
  length = len(matrix)
```

```
  for i in range(length):
    print(f'{i:> {width}}', end=' ')
  print('\n', width * ' ', end='')
  for _ in range(length):
    print('-' * (width + 1), end='')
  for i in range(length):
    print()
    print(f'{i:> {intend}}', end=' |')
    for j in range(length):
        print(f'{matrix[i][j]:> {width}}', end=' ')
  print()

def write_routes(filename, routes):
  with open(filename, 'w') as file:
    writer = csv.writer(file, lineterminator='\n')
    writer.writerow(('start', 'end', 'path'))
    for route in routes:
      writer.writerow((route[0], route[-1], route))
```

Файл config.py

```
n1, n2, n3, n4 = 3, 1, 1, 2

seed = n1 * 1000 + n2 * 100 + n3 * 10 + n4

VERTICES_COUNT = 10 + n3
WIDTH = 800
HEIGHT = 800
LINE_WIDTH = 3
LINE_COLOR = 'white'
ARROWS_LENGTH = 15
VERTEX_RADIUS = 50
SIDES = 4

canvas_options = {
  'bg': 'black',
  'borderwidth': 0,
  'highlightthickness': 0,
}

line_options = {
  'fill': LINE_COLOR,
  'width': LINE_WIDTH,
}

text_options = {
  'fill': LINE_COLOR,
  'font': 14,
}

oval_options = {
  'outline': LINE_COLOR,
  'width': LINE_WIDTH,
}
```

Файл main.py

```
import os
from utils import create_window, zip_vertices
from matrix.print import print_matrix, draw_graph, write_routes
from config import WIDTH, HEIGHT, VERTICES_COUNT, SIDES, n3, n4
from matrix.operations import (
  mul_matrix, transitive_closure, strong_connectivity,
  strong_connectivity_items, condensation_matrix
)
from matrix.graph import(
  adjacency_matrix, to_indirected, vertex_degrees_out, vertex_degrees_in,
routes,
  vertex_degrees, isolated_vertices, hanging_vertices, is_regular
)


def task1(canvas1, canvas2):
  k = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3
  directed = adjacency_matrix(VERTICES_COUNT, k)
  indirected = to_indirected(directed)
  directed_degrees = vertex_degrees(directed, True)
  indirected_degrees = vertex_degrees(indirected, False)
  out = vertex_degrees_out(directed)
  incoming = vertex_degrees_in(directed)
  isolated = isolated_vertices(directed_degrees)
  hanging = hanging_vertices(directed_degrees)
  print('Degree of a vertex in a directed graph:')
  print(*zip_vertices(directed_degrees))
  print('Degree of a vertex in a indirected graph:')
  print(*zip_vertices(indirected_degrees))
  print('Half-degree of the output of the directed graph:')
  print(*zip_vertices(out))
  print('Half-degree of the input of the directed graph:')
  print(*zip_vertices(incoming))
  if is_regular(directed):
```

```
    print('Graph is regular and its degree is', directed_degrees[0])
  else:
    print('Graph isn\'t regular')
  if len(hanging):
    print('Hanging vertices are:')
    print(*zip_vertices(hanging))
  else:
    print('Graph doesn\'t have hanging vertices')
  if len(isolated):
    print('The isolated vertices are:', *isolated)
    print(*zip_vertices(isolated))
  else:
    print('Graph doesn\'t have isolated vertices')
  draw_graph(canvas1, directed, SIDES, True)
  draw_graph(canvas2, indirected, SIDES, False)

def task2(canvas1, canvas2):
  k = 1.0 - n3 * 0.005 - n4 * 0.005 - 0.27
  directed = adjacency_matrix(VERTICES_COUNT, k)
  out = vertex_degrees_out(directed)
  incoming = vertex_degrees_in(directed)
  print('Half-degree of the output of the directed graph:')
  print(*zip_vertices(out))
  print('Half-degree of the input of the directed graph:')
  print(*zip_vertices(incoming))
  directed2 = mul_matrix(directed, directed)
  directed3 = mul_matrix(directed2, directed)
  paths2 = routes(directed, directed2, 2)
  paths3 = routes(directed, directed3, 3)
  write_routes('ruotes2.csv', paths2)
  write_routes('ruotes3.csv', paths3)
  reachability_matrix = transitive_closure(directed, VERTICES_COUNT - 1)
  print('reachability matrix:')
  print_matrix(reachability_matrix)
```

```
    strong_connectivity_matrix = strong_connectivity(reachability_matrix)
    print('strong connectivity matrix:')
    print_matrix(strong_connectivity_matrix)
    print('strong connectivity items:')
    strong_items = strong_connectivity_items(strong_connectivity_matrix)
    print(*strong_items, sep=', ')
    condensation = condensation_matrix(directed, strong_items)
    draw_graph(canvas1, directed, SIDES, True)
    draw_graph(canvas2, condensation, SIDES, True)

if __name__ == '__main__':
    root1, canvas1 = create_window('Task 1 Directed', WIDTH, HEIGHT)
    root2, canvas2 = create_window('Task 1 Indirected', WIDTH, HEIGHT)
    root3, canvas3 = create_window('Task 2 Directed', WIDTH, HEIGHT)
    root4, canvas4 = create_window('Task 2 Condensation', WIDTH, HEIGHT)
    task1(canvas1, canvas2)
    print('—' * os.get_terminal_size().columns)
    task2(canvas3, canvas4)
    root1.mainloop()
    root2.mainloop()
    root3.mainloop()
    root4.mainloop()
```

Файл utils.py

```
from config import canvas_options
import tkinter as tk
import math
import random

float_random = lambda min, max: random.random() * (max - min) + min

minmax = lambda x, y: (min(x, y), max(x, y))

partial = lambda fn, *a, **kw: lambda *p, **k: fn(*a, *p, **kw, **k)

is_neighbors = lambda length, x, y: x == y - 1 or y == length - 1 and x ==
0

def in_one_line(length, ranges, x, y):
  split = math.ceil(length / ranges)
  last_range = length % split
  if last_range == 0:
    last_range = split
  last = length - last_range
  if x == 0 and y >= last and y < length:
    return True
  start = x - x % split
  end = start + split
  return y >= start and y <= end

def create_window(title, width, height):
  root = tk.Tk()
  root.geometry('%dx%d' % (width, height))
  root.title(title)
  canvas = tk.Canvas(root, canvas_options)
  canvas.pack(fill=tk.BOTH, expand=1)
  return (root, canvas)
```

```
def index_filter(arr, fn):
  result = []
  for i, item in enumerate(arr):
    if fn(item):
      result.append(i)
  return result

calculate_step = lambda width, count, sides: (
  width / (math.ceil(count / sides) + 1)
)

zip_vertices = lambda data: ((f'v{i}', n) for i, n in enumerate(data))
```

Файл vertex.py

```
from math import cos, sin, atan2, sqrt, pi, floor, ceil
from dataclasses import dataclass
from utils import calculate_step
from config import (
  line_options, text_options, oval_options,
  ARROWS_LENGTH, VERTEX_RADIUS, WIDTH
)

@dataclass
class Vertex:
  x: int
  y: int
  text: str | int

rotate = lambda x, y, l, f: (
  x + l * cos(f),
  y + l * sin(f),
)

def vertex_arrows(canvas, x, y, fi, delta):
  lx, ly = rotate(x, y, ARROWS_LENGTH, fi + delta)
  rx, ry = rotate(x, y, ARROWS_LENGTH, fi - delta)
  canvas.create_line(lx, ly, x, y, line_options, width=2)
  canvas.create_line(x, y, rx, ry, line_options, width=2)

def vertex_draw(canvas, vertex):
  x, y = vertex.x, vertex.y
  x1, y1 = x + VERTEX_RADIUS, y + VERTEX_RADIUS
  x2, y2 = x - VERTEX_RADIUS, y - VERTEX_RADIUS
  canvas.create_oval(x1, y1, x2, y2, oval_options)
  canvas.create_text(x, y, text=vertex.text, **text_options)

def vertex_line(canvas, v1, v2, shift=False, arrows=False):
```

```
  x1, y1 = v1.x, v1.y
  x2, y2 = v2.x, v2.y
  fi = atan2(y2 - y1, x2 - x1)
  f1 = fi
  f2 = fi + pi
  if shift:
    f1 -= pi / 8
    f2 += pi / 8
  x3, y3 = rotate(x1, y1, VERTEX_RADIUS, f1)
  x4, y4 = rotate(x2, y2, VERTEX_RADIUS, f2)
  canvas.create_line(x3, y3, x4, y4, line_options)
  if arrows:
    vertex_arrows(canvas, x4, y4, fi + pi, pi / 8)

def vertex_arc(canvas, v1, v2, arrows=False):
  fi = atan2(v2.y - v1.y, v2.x - v1.x)
  x3, y3 = rotate(v1.x, v1.y, VERTEX_RADIUS, fi) # - pi / 2
  x4, y4 = rotate(v2.x, v2.y, VERTEX_RADIUS, fi + pi) # + pi / 2
  meadle = sqrt((x4 - x3)**2 + (y4 - y3)**2) / 2
  length = sqrt((3 * VERTEX_RADIUS)**2 + meadle**2)
  f = -atan2(3 * VERTEX_RADIUS, meadle)
  x5, y5 = rotate(x3, y3, length, f + fi)
  canvas.create_line(x3, y3, x5, y5, x4, y4, line_options, smooth=1)
  if arrows:
    vertex_arrows(canvas, x4, y4, fi + pi - f, pi / 8)

def vertex_loop(canvas, vertex, arrows=False):
  x = vertex.x
  y = vertex.y - VERTEX_RADIUS
  x1, y1 = rotate(x, y, VERTEX_RADIUS, -pi / 4)
  x2, y2 = rotate(x, y, VERTEX_RADIUS, -3 * pi / 4)
  canvas.create_line(x, y, x1, y1, x2, y2, x, y, line_options)
  if arrows:
    vertex_arrows(canvas, x, y, pi / 4 + pi, pi / 8)
```

```
cases = (
  lambda i, sp, st, start: (start + st * i, start),
  lambda i, sp, st, start: (start + st * sp, start + st * i),
  lambda i, sp, st, start: (start + st * (sp - i), start + st * sp),
  lambda i, sp, st, start: (start, start + st * (sp - i)),
)

def get_vertex_closure(length, sides):
  split = ceil(length / sides)
  step = calculate_step(WIDTH, length, sides)
  start = step / 2
  def wrapped(index):
    side = floor(index / split)
    x, y = cases[side](index % split, split, step, start)
    return Vertex(x, y, index)
  return wrapped
```

За п. 1 завдання: згенеровані матриці суміжності напрямленого та ненапрямленого графів.

Матриця суміжності напрямленого графу:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	1	1	0	0	1
1	0	0	0	0	0	0	0	0	0	1	1
2	0	0	1	0	0	1	1	1	0	0	1
3	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	1	0	0	0	0
5	0	1	0	1	1	0	1	0	0	1	0
6	0	1	0	0	0	0	0	1	0	1	0
7	1	0	1	0	0	0	0	0	1	1	0
8	1	0	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0	0

Матриця суміжності ненапрямленого графу:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	1	1	1	1	1
1	0	0	0	0	0	1	1	0	0	1	1
2	0	0	1	0	0	1	1	1	0	0	1
3	1	0	0	1	1	1	0	0	0	0	0
4	0	0	0	1	0	1	1	0	1	0	0
5	0	1	1	1	1	0	1	0	0	1	0
6	1	1	1	0	1	1	0	1	0	1	0
7	1	0	1	0	0	0	1	0	1	1	0
8	1	0	0	0	1	0	0	1	0	1	0
9	1	1	0	0	0	1	1	1	1	0	0
10	1	1	1	0	0	0	0	0	0	0	0

За п. 2: перелік степенів, півстепенів, результат перевірки на однорідність, переліки висячих та ізольованих вершин.

```
Degree of a vertex in a directed graph:
('v0', 7) ('v1', 4) ('v2', 7) ('v3', 5) ('v4', 4) ('v5', 6) ('v6', 7)
('v7', 7) ('v8', 4) ('v9', 6) ('v10', 3)
Degree of a vertex in a indirected graph:
('v0', 6) ('v1', 4) ('v2', 6) ('v3', 5) ('v4', 4) ('v5', 6) ('v6', 7)
('v7', 5) ('v8', 4) ('v9', 6) ('v10', 3)
Half-degree of the output of the directed graph:
('v0', 4) ('v1', 2) ('v2', 5) ('v3', 1) ('v4', 2) ('v5', 5) ('v6', 3)
('v7', 4) ('v8', 2) ('v9', 2) ('v10', 0)
Half-degree of the input of the directed graph:
('v0', 3) ('v1', 2) ('v2', 2) ('v3', 4) ('v4', 2) ('v5', 1) ('v6', 4)
('v7', 3) ('v8', 2) ('v9', 4) ('v10', 3)
Graph isn't regular
Graph doesn't have hanging vertices
Graph doesn't have isolated vertices
```

За п. 3: матриця другого орграфа.

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	1	1	0	0	1
1	0	0	0	0	0	1	0	0	0	1	1
2	0	1	1	0	0	1	1	1	0	0	1
3	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	1	0	0	0	0
5	0	1	0	1	1	0	1	0	0	1	0
6	0	1	0	0	1	0	0	1	0	1	0
7	1	0	1	0	0	0	0	0	1	1	0
8	1	0	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	1	0	0
10	1	0	0	0	0	0	0	0	0	0	0

За п. 4: переліки півстепенів, шляхів, матриці досяжності та сильної зв'язності, перелік компонент сильної зв'язності, граф конденсації.

Переліки півстепенів:

```
Half-degree of the output of the directed graph:
('v0', 4) ('v1', 3) ('v2', 6) ('v3', 1) ('v4', 2) ('v5', 5) ('v6', 4)
('v7', 4) ('v8', 2) ('v9', 2) ('v10', 1)
Half-degree of the input of the directed graph:
('v0', 4) ('v1', 3) ('v2', 2) ('v3', 4) ('v4', 3) ('v5', 2) ('v6', 4)
('v7', 3) ('v8', 2) ('v9', 4) ('v10', 3)
```

Всі шляхи довжиною 2:

start	end	path	start	end	path	start	end	path
0	0	(0, 7, 0)	2	8	(2, 7, 8)	6	9	(6, 7, 9)
0	0	(0, 10, 0)	2	9	(2, 1, 9)	6	10	(6, 1, 10)
0	1	(0, 6, 1)	2	9	(2, 5, 9)	7	0	(7, 8, 0)
0	2	(0, 7, 2)	2	9	(2, 6, 9)	7	0	(7, 9, 0)
0	3	(0, 3, 3)	2	9	(2, 7, 9)	7	1	(7, 2, 1)
0	4	(0, 6, 4)	2	10	(2, 1, 10)	7	2	(7, 2, 2)
0	7	(0, 6, 7)	2	10	(2, 2, 10)	7	3	(7, 0, 3)
0	8	(0, 7, 8)	4	1	(4, 6, 1)	7	4	(7, 8, 4)
0	9	(0, 6, 9)	4	3	(4, 3, 3)	7	5	(7, 2, 5)
0	9	(0, 7, 9)	4	4	(4, 6, 4)	7	6	(7, 0, 6)
1	0	(1, 9, 0)	4	7	(4, 6, 7)	7	6	(7, 2, 6)
1	0	(1, 10, 0)	4	9	(4, 6, 9)	7	7	(7, 0, 7)
1	1	(1, 5, 1)	5	0	(5, 9, 0)	7	7	(7, 2, 7)
1	3	(1, 5, 3)	5	1	(5, 6, 1)	7	8	(7, 9, 8)
1	4	(1, 5, 4)	5	3	(5, 3, 3)	7	10	(7, 0, 10)
1	6	(1, 5, 6)	5	3	(5, 4, 3)	7	10	(7, 2, 10)
1	8	(1, 9, 8)	5	4	(5, 6, 4)	8	3	(8, 0, 3)
1	9	(1, 5, 9)	5	5	(5, 1, 5)	8	3	(8, 4, 3)
2	0	(2, 7, 0)	5	6	(5, 4, 6)	8	6	(8, 0, 6)
2	0	(2, 10, 0)	5	7	(5, 6, 7)	8	6	(8, 4, 6)
2	1	(2, 2, 1)	5	8	(5, 9, 8)	8	7	(8, 0, 7)
2	1	(2, 5, 1)	5	9	(5, 1, 9)	8	10	(8, 0, 10)
2	1	(2, 6, 1)	5	9	(5, 6, 9)	9	0	(9, 8, 0)
2	2	(2, 7, 2)	5	10	(5, 1, 10)	9	3	(9, 0, 3)
2	3	(2, 5, 3)	6	0	(6, 7, 0)	9	4	(9, 8, 4)
2	4	(2, 5, 4)	6	0	(6, 9, 0)	9	6	(9, 0, 6)
2	4	(2, 6, 4)	6	2	(6, 7, 2)	9	7	(9, 0, 7)
2	5	(2, 1, 5)	6	3	(6, 4, 3)	9	10	(9, 0, 10)
2	5	(2, 2, 5)	6	5	(6, 1, 5)	10	3	(10, 0, 3)
2	6	(2, 2, 6)	6	6	(6, 4, 6)	10	6	(10, 0, 6)
2	6	(2, 5, 6)	6	8	(6, 7, 8)	10	7	(10, 0, 7)
2	7	(2, 2, 7)	6	8	(6, 9, 8)	10	10	(10, 0, 10)
2	7	(2, 6, 7)	6	9	(6, 1, 9)			

Всі шляхи довжиною 3:

start	end	path	start	end	path	start	end	path
0	0	(0, 6, 7, 0)	2	7	(2, 5, 6, 7)	6	8	(6, 7, 9, 8)
0	0	(0, 6, 9, 0)	2	7	(2, 7, 0, 7)	6	9	(6, 1, 5, 9)
0	0	(0, 7, 8, 0)	2	7	(2, 10, 0, 7)	6	9	(6, 4, 6, 9)
0	0	(0, 7, 9, 0)	2	8	(2, 1, 9, 8)	6	10	(6, 7, 0, 10)
0	1	(0, 7, 2, 1)	2	8	(2, 2, 7, 8)	6	10	(6, 7, 2, 10)
0	2	(0, 6, 7, 2)	2	8	(2, 5, 9, 8)	6	10	(6, 9, 0, 10)
0	2	(0, 7, 2, 2)	2	8	(2, 6, 7, 8)	7	0	(7, 0, 10, 0)
0	3	(0, 6, 4, 3)	2	8	(2, 6, 9, 8)	7	0	(7, 2, 7, 0)
0	3	(0, 7, 0, 3)	2	8	(2, 7, 9, 8)	7	0	(7, 2, 10, 0)
0	3	(0, 10, 0, 3)	2	9	(2, 1, 5, 9)	7	0	(7, 9, 8, 0)
0	4	(0, 7, 8, 4)	2	9	(2, 2, 1, 9)	7	1	(7, 0, 6, 1)
0	5	(0, 6, 1, 5)	2	9	(2, 2, 5, 9)	7	1	(7, 2, 2, 1)
0	5	(0, 7, 2, 5)	2	9	(2, 2, 6, 9)	7	1	(7, 2, 5, 1)
0	6	(0, 6, 4, 6)	2	9	(2, 2, 7, 9)	7	1	(7, 2, 6, 1)
0	6	(0, 7, 0, 6)	2	9	(2, 5, 1, 9)	7	2	(7, 0, 7, 2)
0	6	(0, 7, 2, 6)	2	9	(2, 5, 6, 9)	7	3	(7, 0, 3, 3)
0	6	(0, 10, 0, 6)	2	9	(2, 6, 1, 9)	7	3	(7, 2, 5, 3)
0	7	(0, 7, 2, 7)	2	9	(2, 6, 7, 9)	7	3	(7, 8, 0, 3)
0	7	(0, 10, 0, 7)	2	10	(2, 2, 1, 10)	7	3	(7, 8, 4, 3)
0	8	(0, 6, 7, 8)	2	10	(2, 5, 1, 10)	7	3	(7, 9, 0, 3)
0	8	(0, 6, 9, 8)	2	10	(2, 6, 1, 10)	7	4	(7, 0, 6, 4)
0	8	(0, 7, 9, 8)	2	10	(2, 7, 0, 10)	7	4	(7, 2, 5, 4)
0	9	(0, 6, 1, 9)	2	10	(2, 7, 2, 10)	7	4	(7, 2, 6, 4)
0	9	(0, 6, 7, 9)	2	10	(2, 10, 0, 10)	7	4	(7, 9, 8, 4)
0	10	(0, 6, 1, 10)	4	0	(4, 6, 7, 0)	7	5	(7, 2, 1, 5)
0	10	(0, 7, 0, 10)	4	0	(4, 6, 9, 0)	7	5	(7, 2, 2, 5)
0	10	(0, 7, 2, 10)	4	2	(4, 6, 7, 2)	7	6	(7, 2, 2, 6)
1	0	(1, 5, 9, 0)	4	3	(4, 6, 4, 3)	7	6	(7, 2, 5, 6)
1	0	(1, 9, 8, 0)	4	5	(4, 6, 1, 5)	7	6	(7, 8, 0, 6)
1	1	(1, 5, 6, 1)	4	8	(4, 6, 7, 8)	7	6	(7, 8, 4, 6)
1	3	(1, 5, 3, 3)	4	8	(4, 6, 9, 8)	7	6	(7, 9, 0, 6)
1	3	(1, 5, 4, 3)	4	9	(4, 6, 1, 9)	7	7	(7, 0, 6, 7)
1	3	(1, 9, 0, 3)	4	9	(4, 6, 7, 9)	7	7	(7, 2, 2, 7)
1	3	(1, 10, 0, 3)	4	10	(4, 6, 1, 10)	7	7	(7, 2, 6, 7)
1	4	(1, 5, 6, 4)	5	0	(5, 1, 9, 0)	7	7	(7, 8, 0, 7)
1	4	(1, 9, 8, 4)	5	0	(5, 1, 10, 0)	7	7	(7, 9, 0, 7)
1	6	(1, 5, 4, 6)	5	0	(5, 6, 7, 0)	7	8	(7, 0, 7, 8)
1	6	(1, 9, 0, 6)	5	0	(5, 6, 9, 0)	7	8	(7, 2, 7, 8)
1	6	(1, 10, 0, 6)	5	0	(5, 9, 8, 0)	7	9	(7, 0, 6, 9)
1	7	(1, 5, 6, 7)	5	1	(5, 4, 6, 1)	7	9	(7, 0, 7, 9)
1	7	(1, 9, 0, 7)	5	2	(5, 6, 7, 2)	7	9	(7, 2, 1, 9)
1	7	(1, 10, 0, 7)	5	3	(5, 1, 5, 3)	7	9	(7, 2, 5, 9)
1	8	(1, 5, 9, 8)	5	3	(5, 4, 3, 3)	7	9	(7, 2, 6, 9)

1	9	(1, 5, 1, 9)	5	3	(5, 6, 4, 3)	7	9	(7, 2, 7, 9)
1	9	(1, 5, 6, 9)	5	3	(5, 9, 0, 3)	7	10	(7, 2, 1, 10)
1	10	(1, 5, 1, 10)	5	4	(5, 1, 5, 4)	7	10	(7, 2, 2, 10)
1	10	(1, 9, 0, 10)	5	4	(5, 4, 6, 4)	7	10	(7, 8, 0, 10)
1	10	(1, 10, 0, 10)	5	4	(5, 9, 8, 4)	7	10	(7, 9, 0, 10)
2	0	(2, 1, 9, 0)	5	5	(5, 6, 1, 5)	8	0	(8, 0, 7, 0)
2	0	(2, 1, 10, 0)	5	6	(5, 1, 5, 6)	8	0	(8, 0, 10, 0)
2	0	(2, 2, 7, 0)	5	6	(5, 6, 4, 6)	8	1	(8, 0, 6, 1)
2	0	(2, 2, 10, 0)	5	6	(5, 9, 0, 6)	8	1	(8, 4, 6, 1)
2	0	(2, 5, 9, 0)	5	7	(5, 4, 6, 7)	8	2	(8, 0, 7, 2)
2	0	(2, 6, 7, 0)	5	7	(5, 9, 0, 7)	8	3	(8, 0, 3, 3)
2	0	(2, 6, 9, 0)	5	8	(5, 1, 9, 8)	8	3	(8, 4, 3, 3)
2	0	(2, 7, 8, 0)	5	8	(5, 6, 7, 8)	8	4	(8, 0, 6, 4)
2	0	(2, 7, 9, 0)	5	8	(5, 6, 9, 8)	8	4	(8, 4, 6, 4)
2	1	(2, 1, 5, 1)	5	9	(5, 1, 5, 9)	8	7	(8, 0, 6, 7)
2	1	(2, 2, 5, 1)	5	9	(5, 4, 6, 9)	8	7	(8, 4, 6, 7)
2	1	(2, 2, 6, 1)	5	9	(5, 6, 1, 9)	8	8	(8, 0, 7, 8)
2	1	(2, 5, 6, 1)	5	9	(5, 6, 7, 9)	8	9	(8, 0, 6, 9)
2	1	(2, 7, 2, 1)	5	10	(5, 6, 1, 10)	8	9	(8, 0, 7, 9)
2	2	(2, 2, 7, 2)	5	10	(5, 9, 0, 10)	8	9	(8, 4, 6, 9)
2	2	(2, 6, 7, 2)	6	0	(6, 1, 9, 0)	9	0	(9, 0, 7, 0)
2	2	(2, 7, 2, 2)	6	0	(6, 1, 10, 0)	9	0	(9, 0, 10, 0)
2	3	(2, 1, 5, 3)	6	0	(6, 7, 8, 0)	9	1	(9, 0, 6, 1)
2	3	(2, 2, 5, 3)	6	0	(6, 7, 9, 0)	9	2	(9, 0, 7, 2)
2	3	(2, 5, 3, 3)	6	0	(6, 9, 8, 0)	9	3	(9, 0, 3, 3)
2	3	(2, 5, 4, 3)	6	1	(6, 1, 5, 1)	9	3	(9, 8, 0, 3)
2	3	(2, 6, 4, 3)	6	1	(6, 4, 6, 1)	9	3	(9, 8, 4, 3)
2	3	(2, 7, 0, 3)	6	1	(6, 7, 2, 1)	9	4	(9, 0, 6, 4)
2	3	(2, 10, 0, 3)	6	2	(6, 7, 2, 2)	9	6	(9, 8, 0, 6)
2	4	(2, 1, 5, 4)	6	3	(6, 1, 5, 3)	9	6	(9, 8, 4, 6)
2	4	(2, 2, 5, 4)	6	3	(6, 4, 3, 3)	9	7	(9, 0, 6, 7)
2	4	(2, 2, 6, 4)	6	3	(6, 7, 0, 3)	9	7	(9, 8, 0, 7)
2	4	(2, 5, 6, 4)	6	3	(6, 9, 0, 3)	9	8	(9, 0, 7, 8)
2	4	(2, 7, 8, 4)	6	4	(6, 1, 5, 4)	9	9	(9, 0, 6, 9)
2	5	(2, 2, 1, 5)	6	4	(6, 7, 8, 4)	9	9	(9, 0, 7, 9)
2	5	(2, 5, 1, 5)	6	4	(6, 9, 8, 4)	9	10	(9, 8, 0, 10)
2	5	(2, 6, 1, 5)	6	5	(6, 7, 2, 5)	10	0	(10, 0, 7, 0)
2	5	(2, 7, 2, 5)	6	6	(6, 1, 5, 6)	10	1	(10, 0, 6, 1)
2	6	(2, 1, 5, 6)	6	6	(6, 7, 0, 6)	10	2	(10, 0, 7, 2)
2	6	(2, 2, 5, 6)	6	6	(6, 7, 2, 6)	10	3	(10, 0, 3, 3)
2	6	(2, 5, 4, 6)	6	6	(6, 9, 0, 6)	10	4	(10, 0, 6, 4)
2	6	(2, 6, 4, 6)	6	7	(6, 4, 6, 7)	10	7	(10, 0, 6, 7)
2	6	(2, 7, 0, 6)	6	7	(6, 7, 0, 7)	10	8	(10, 0, 7, 8)
2	6	(2, 7, 2, 6)	6	7	(6, 7, 2, 7)	10	9	(10, 0, 6, 9)
2	6	(2, 10, 0, 6)	6	7	(6, 9, 0, 7)	10	9	(10, 0, 7, 9)

2	7	(2, 2, 6, 7)	6	8	(6, 1, 9, 8)			

Матриця досяжності:

	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	1	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1	1

Матриця сильної зв’язності:

	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	0	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1
2	1	1	1	0	1	1	1	1	1	1	1
3	0	0	0	1	0	0	0	0	0	0	0
4	1	1	1	0	1	1	1	1	1	1	1
5	1	1	1	0	1	1	1	1	1	1	1
6	1	1	1	0	1	1	1	1	1	1	1
7	1	1	1	0	1	1	1	1	1	1	1
8	1	1	1	0	1	1	1	1	1	1	1
9	1	1	1	0	1	1	1	1	1	1	1
10	1	1	1	0	1	1	1	1	1	1	1

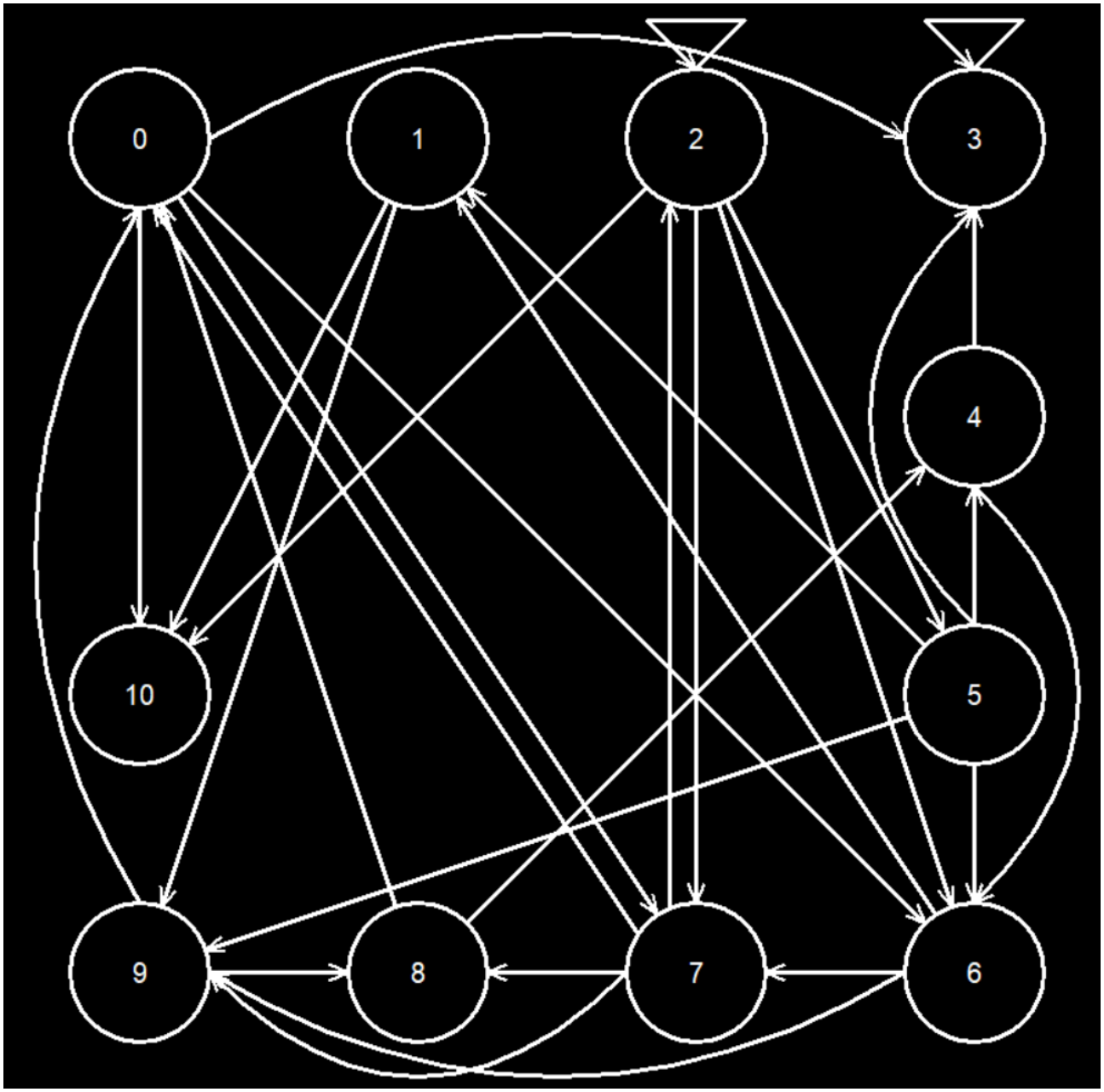
Елементи сильної зв’язності:

```
[0, 1, 2, 4, 5, 6, 7, 8, 9, 10], [3]
```

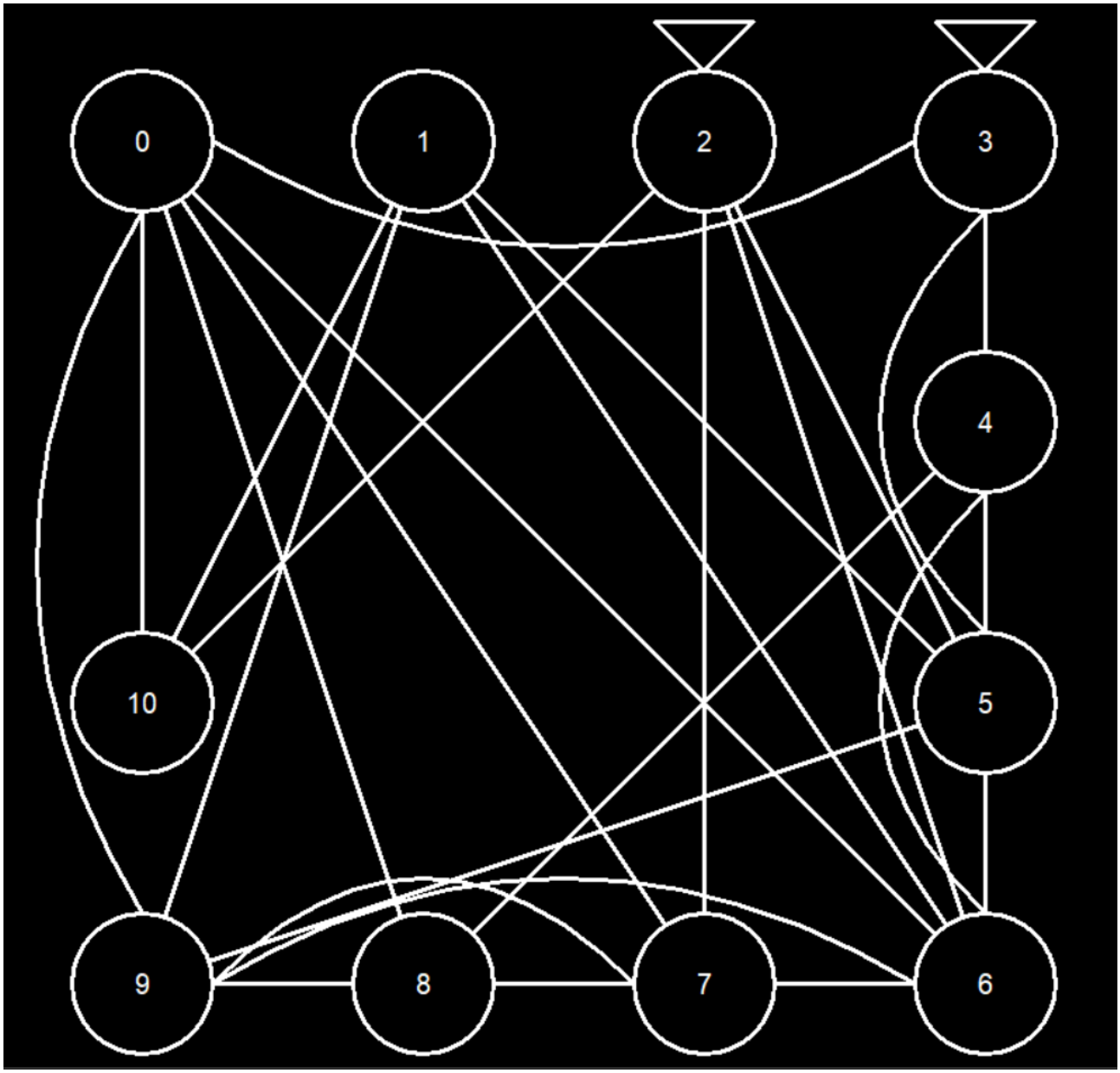
Граф конденсації:

Скришити заданих орієнтованого і неорієнтованого графів, модифікованого графа та його графа конденсації.

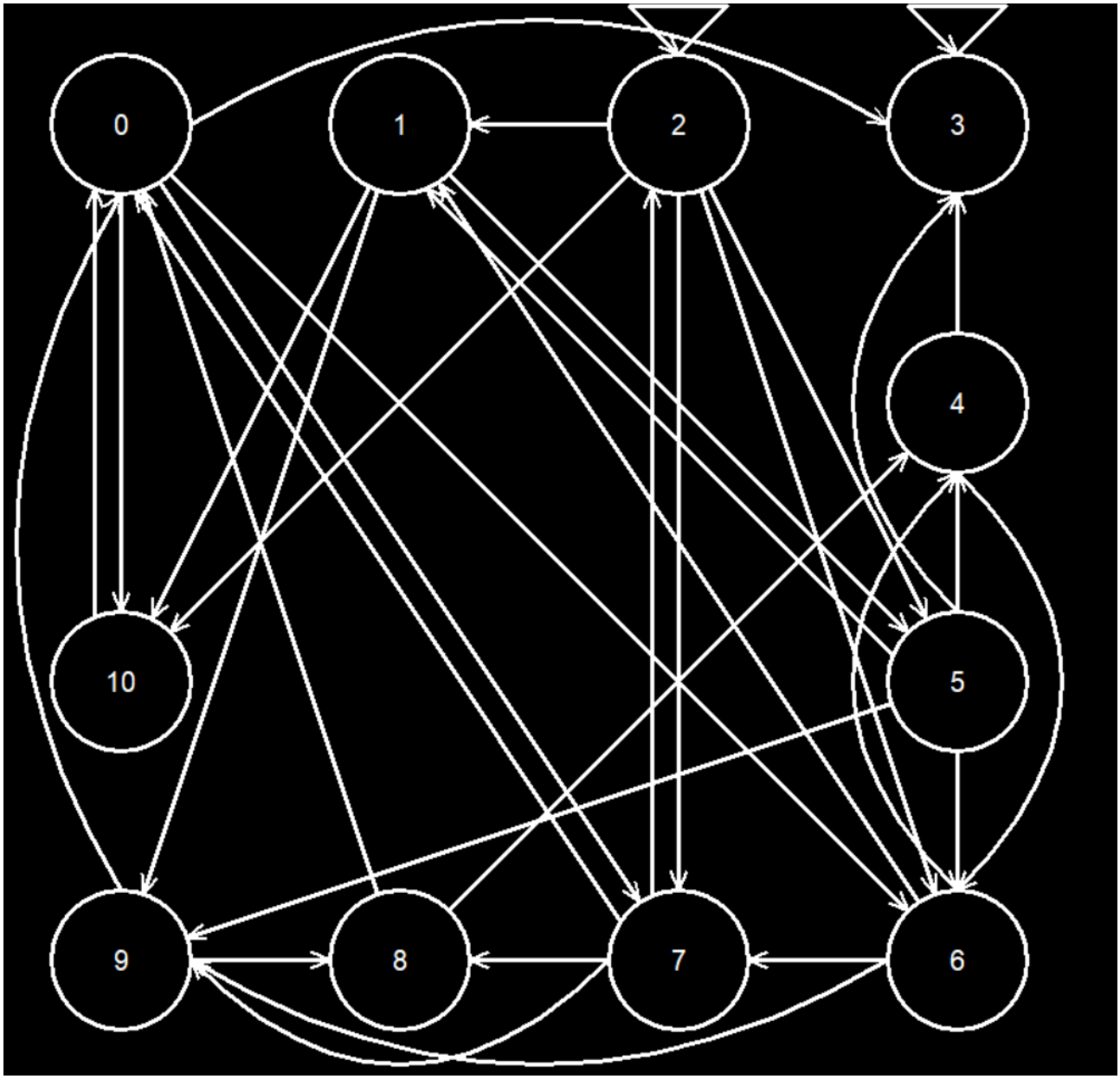
Орієнтований початковий граф:



Неорієнтований початковий граф:



Модифікований граф:



Граф конденсації:

Висновки

Протягом виконання лабораторної роботи я дослідив характеристики графів та навчився визначати їх на конкретних прикладах, на практиці засвоїв метод транзитивного замикання. Я написав програму, яка графічно відображає граф за матрицею суміжності та обчислює деякі його характеристики. Також я набув практичних навичок в програмуванні алгоритмів операцій над матрицями, таких як множення, додавання, транспонування та інші. Закріпив на практиці знання з дискретної математики про відношення, а саме композицію, транзитивне замикання, об’єднання і т. д. Також був розроблений алгоритм для пошуку усіх шляхів між двома вершинами за відомою довжиною. Я узагальнив його для всіх випадків, тож за допомогою розробленої функції можна шукати всі шляхи не лише довжини 2 та 3, як вказано у завданні, а й будь-якої іншої.

Як результат виконання лабораторної роботи, я закріпив знання про графи, їх структуру та властивості, навчився знаходити деякі характеристики графів та покращив навички в програмуванні алгоритмів.