**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 №5**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірила:

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

Київ 2024

**Постановка задачі**

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

**Відмінність**: коефіцієнт k = 1.0 – n3 \* 0.01 – n4 \* 0.05 – 0.15.

Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1. встановлюється параметр (seed) генератора випадкових чисел, рiвний номеру варiанту n1n2n3n4;
2. матриця розмiром n \* n заповнюється згенерованими випадковими числами в дiапазонi [0, 2.0);
3. обчислюється коефiцiєнт k = 1.0 – n3 \* 0.01 – n4 \* 0.05 – 0.15, кожен елемент матрицi множиться на коефiцiєнт k;
4. елементи матрицi округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.
5. Створити програму, яка виконує обхiд напрямленого графа вшир (BFS) та вглиб (DFS).

* обхiд починати з вершини iз найменшим номером, яка має щонайменше одну вихiдну дугу;
* при обходi враховувати порядок нумерацiї;
* у програмi виконання обходу вiдображати покроково, черговий крок виконувати за натисканням кнопки у вiкнi або на клавiатурi.

**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: згенерованi матрицi сумiжностi напрямленого та ненапрямленого графiв.**

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: перелiк степенів, пiвстепенiв, 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: перелiки пiвстепенiв, шляхiв, матрицi досяжностi та сильної зв’язностi, перелiк компонент сильної зв’язностi, граф конденсацiї.**

Перелiки пiвстепенiв:

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

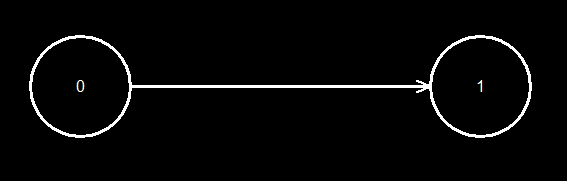
9 | 1 1 1 0 1 1 1 1 1 1 1

10 | 1 1 1 0 1 1 1 1 1 1 1

Елементи сильної зв’язності:

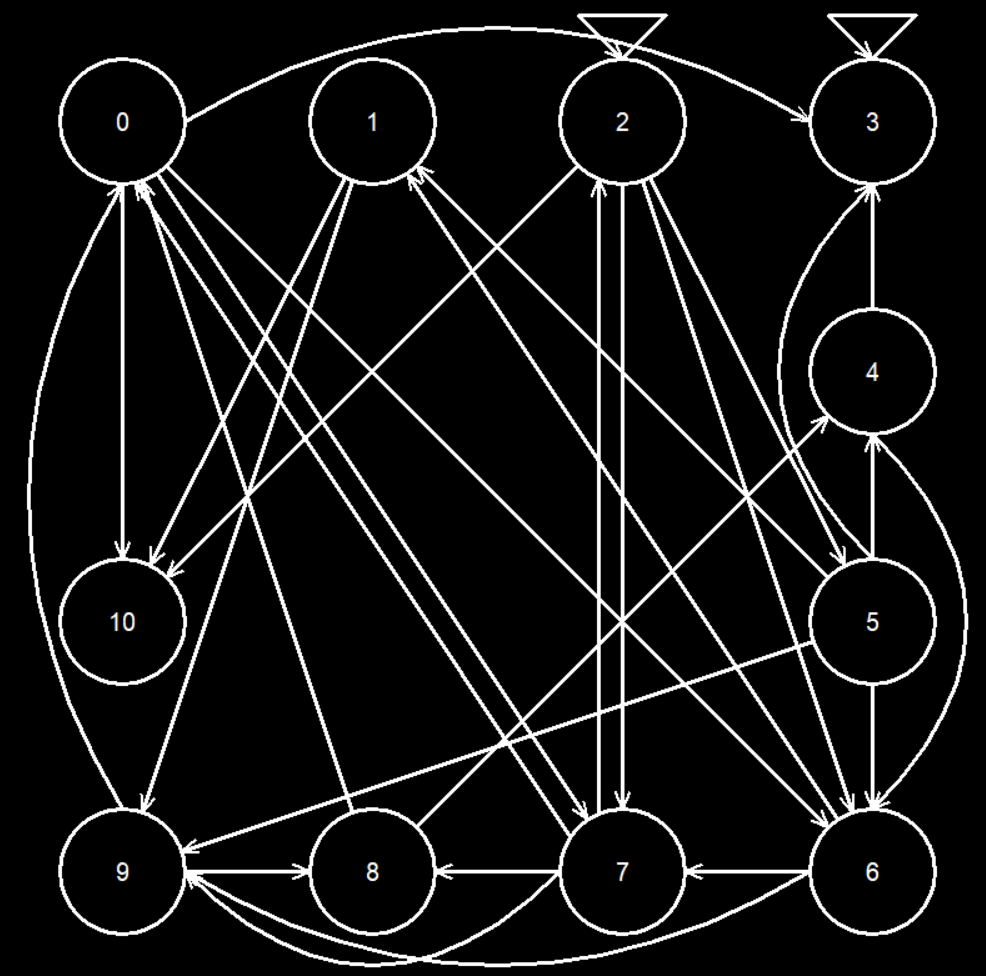
[0, 1, 2, 4, 5, 6, 7, 8, 9, 10], [3]

Граф конденсації:

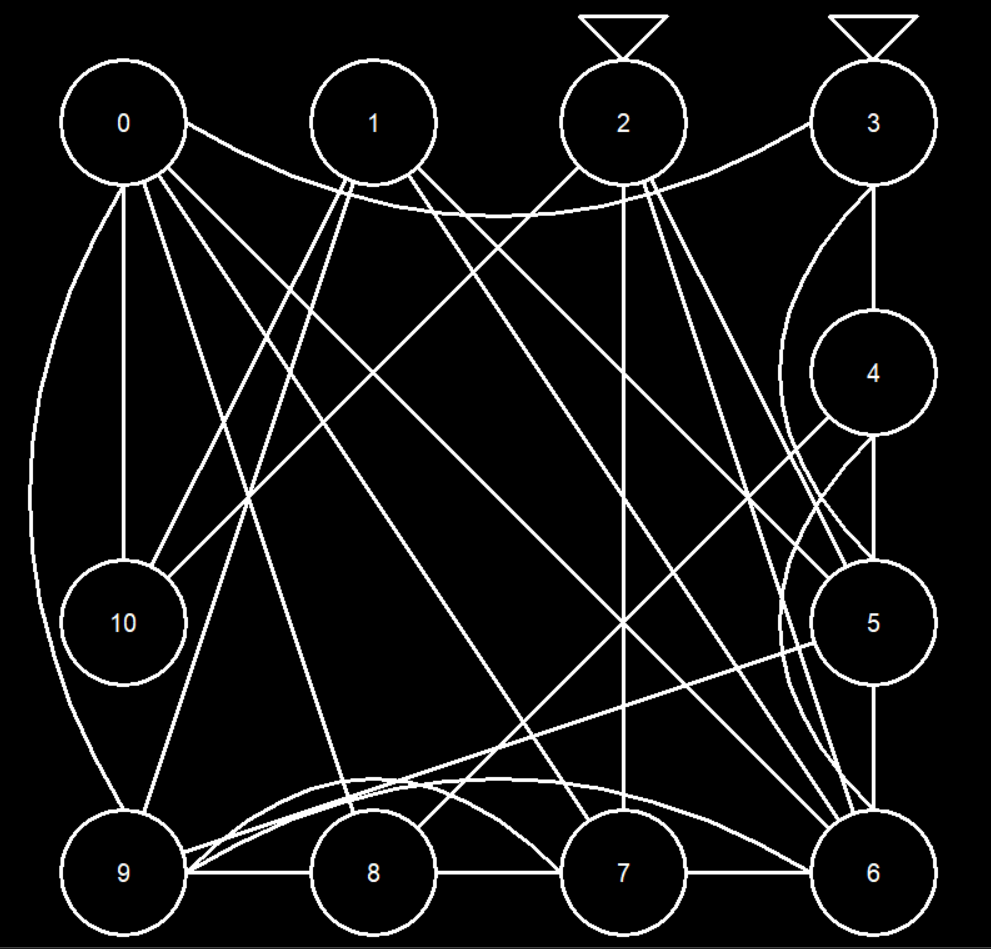


**Скришити заданих орієнтованого i неорієнтованого графiв, модифікованого графа та його графа конденсацiї.**

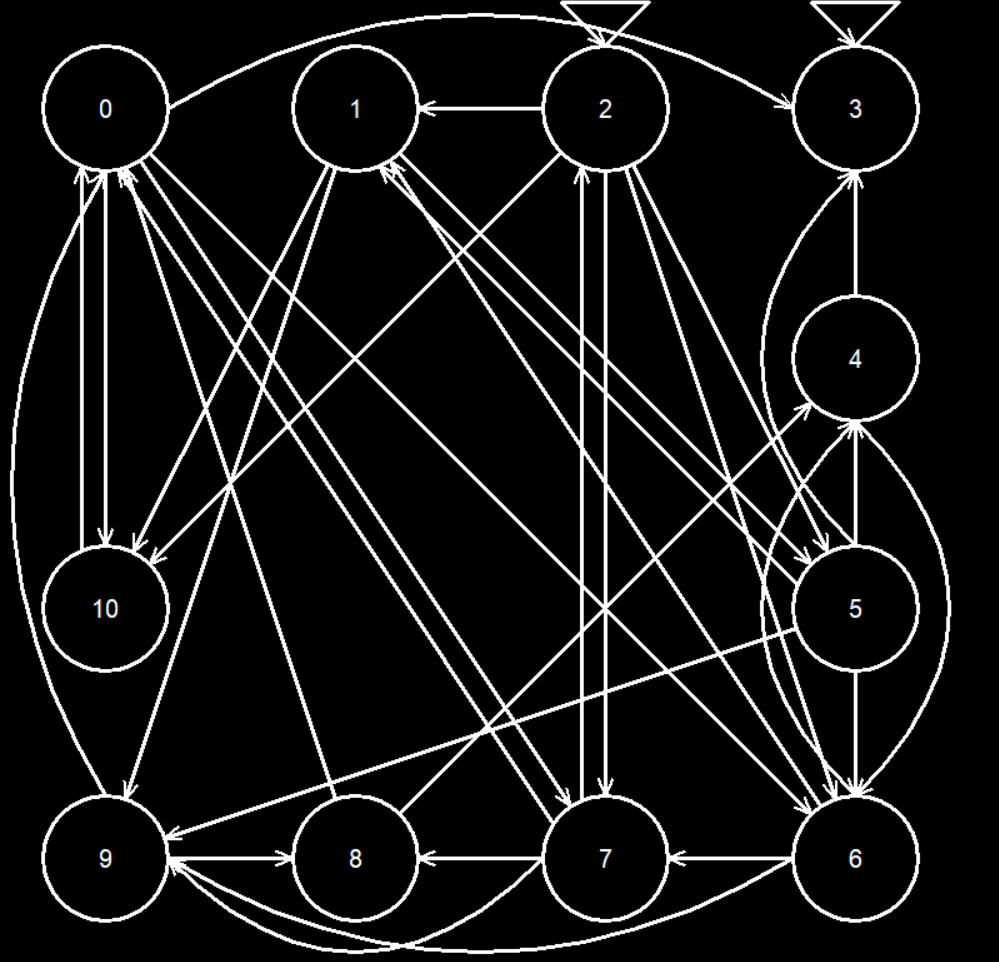
Орієнтований початковий граф:



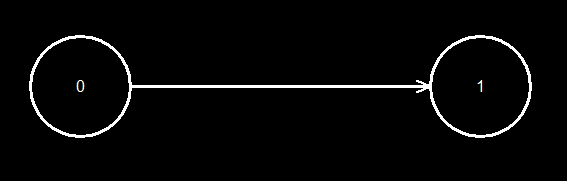
Неорієнтований початковий граф:



Модифікований граф:



Граф конденсації:



**Висновки**

Протягом виконання лабораторної роботи я дослідив характеристики графiв 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.