**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: Перевірила:

студент групи ІМ-44 Сергієнко М. А.  
Мундурс Нікіта Юрійович  
номер у списку групи: 16

Київ 2025

**Завдання**

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

***Відмінність***: коефіцієнт *k* = 1.0 – *n3* \* 0.01 - *n4* \* 0.01 – 0.3

Матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1. встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту *n1n2n3n4*;
2. матриця розміром *n*\**n* заповнюється згенерованими випадковими числами в діапазоні [0, 2,0);
3. обчислюється коефіцієнт *k* = 1.0 – *n3* \* 0.01 - *n4* \* 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.
5. Обчислити:
6. степені вершин напрямленого і ненапрямленого графів;
7. напівстепені виходу та заходу напрямленого графа;
8. чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;
9. перелік висячих та ізольованих вершин.

Результати вивести в графічне вікно, консоль або файл.

1. Змінити матрицю Adir, коефіцієнт *k* = 1.0 – *n3* \* 0.005 - *n4* \* 0.005 – 0.27
2. Для нового орграфа обчислити:
3. півстепені вершин;
4. всі шляхи довжини 2 і 3;
5. матрицю досяжності;
6. матрицю сильної зв’язності;
7. перелік компонент сильної зв’язності;
8. граф конденсації.

Результати вивести в графічне вікно, 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 в графічне вікно.

При проєктуваннi програм ***слiд врахувати наступне***:

1. мова програмування обирається студентом самостійно;
2. графічне зображення всіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;
3. усі графи, включно із графом конденсації, обов’язково зображувати в графічному вікні;
4. типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;
5. обчислення перелічених у завданні результатів має виконуватися розробленою програмою (не вручну і не сторонніми засобами);
6. матриці, переліки степенів та маршрутів тощо можна виводити в графічне вікно або консоль – на розсуд студента;
7. у переліку знайдених шляхів треба вказувати не лише початок та кінець шляху, але й усі проміжні вершини, через які він проходить (наприклад, 1 – 5 – 3 – 2).

**Варіант:** 4416

***k1* =** 0,63

***k2* =** 0,695

**Кількість вершин:** 11

**Розміщення вершин:** колом з вершиною в центрі

**Обрана мова програмування**: Python

**Тексти програм**

*Програма для виконання пунктів 1, 2; ненапрямлений граф*

import turtle

import random

import math

class Vertex:

def \_\_init\_\_(self, number, pos\_x, pos\_y):

self.number = number

self.pos\_x = pos\_x

self.pos\_y = pos\_y

def main():

seed = 4416

n3 = 1

n4 = 6

N = 11

directed\_graph\_matrix = generateMatrix(n3, n4, N, seed)

undirected\_graph\_matrix = doUndir(directed\_graph\_matrix)

position = createPositions(undirected\_graph\_matrix)

degree\_of\_vertex = getDegreeOfVertex(undirected\_graph\_matrix)

checkForUniformity(degree\_of\_vertex)

findFinalVertices(degree\_of\_vertex)

findIsolatedVertices(degree\_of\_vertex)

drawVertices(position)

drawLines(position, undirected\_graph\_matrix)

drawCurvedLines(position, undirected\_graph\_matrix)

drawCircles(position, undirected\_graph\_matrix)

def generateMatrix(n3, n4, N, seed):

random.seed(seed)

adj\_matrix = [[random.random() \* 2.0 for \_ in range(N)] for \_ in range(N)]

k = 1.0 - n3 \* 0.01 - n4 \* 0.01 - 0.3

for i in range(N):

for j in range(N):

adj\_matrix[i][j] \*= k

adj\_matrix[i][j] = 0 if adj\_matrix[i][j] < 1.0 else 1

return adj\_matrix

def doUndir(adj\_matrix):

n = len(adj\_matrix)

undirected\_adj\_matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

if adj\_matrix[i][j] == 1:

undirected\_adj\_matrix[i][j] = 1

undirected\_adj\_matrix[j][i] = 1

print("\nМатриця суміжності ненапрямленого графа:")

for row in undirected\_adj\_matrix:

print(row)

return undirected\_adj\_matrix

def createPositions(undirected\_graph\_matrix):

n = len(undirected\_graph\_matrix)

vertices = []

center\_x = 0

center\_y = 0

radius = 300

for i in range(1, n):

angle = i \* (2 \* math.pi / (n - 1))

x = center\_x + radius \* math.cos(angle)

y = center\_y + radius \* math.sin(angle)

vertices.append(Vertex(i, x, y))

central\_vertex = Vertex(n, center\_x, center\_y)

vertices.append(central\_vertex)

return vertices

def drawVertices(vertices):

turtle.speed(0)

turtle.penup()

radius = 20

for vertex in vertices:

x, y = vertex.pos\_x, vertex.pos\_y

turtle.goto(x, y - radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x, y)

turtle.write(vertex.number, align="center")

def drawLines(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

for i in range(n):

for j in range(i + 1, n):

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) != 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

angle = math.atan2(y2 - y1, x2 - x1)

x1 += math.cos(angle) \* radius

y1 += math.sin(angle) \* radius

x2 -= math.cos(angle) \* radius

y2 -= math.sin(angle) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.pendown()

turtle.goto(x2, y2)

def drawCurvedLines(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

for i in range(n):

for j in range(i + 1, n):

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) == 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

angle = math.atan2(y2 - y1, x2 - x1)

x1 += math.cos(angle) \* radius

y1 += math.sin(angle) \* radius

x2 -= math.cos(angle) \* radius

y2 -= math.sin(angle) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(math.degrees(angle))

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = math.pi \* 2

b = distance / 2 / math.cos(math.radians(degrees))

turtle.right(degrees)

turtle.forward(b)

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

def drawCircles(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 10

for i in range(n):

if undirected\_graph\_matrix[i][i] == 1:

x, y = vertices[i].pos\_x, vertices[i].pos\_y

turtle.goto(x + math.pi / 2.8 \* radius, y + math.pi / 2 \* radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.hideturtle()

turtle.done()

def getDegreeOfVertex(adjacency\_matrix):

num\_vertices = len(adjacency\_matrix)

edges = {}

for i in range(1, num\_vertices + 1):

edges[i] = sum(adjacency\_matrix[i - 1])

print("Степені вершин:", edges)

return edges

def checkForUniformity(degree\_of\_vertex):

edges\_counts = list(degree\_of\_vertex.values())

if len(set(edges\_counts)) == 1:

print("Граф однорідний. Степінь однорідності:", edges\_counts[0])

else:

print("Граф не є однорідним")

def findFinalVertices(degree\_of\_vertex):

vertices\_with\_one\_edge = [v for v, deg in degree\_of\_vertex.items() if deg == 1]

if vertices\_with\_one\_edge:

print("Висячі вершини:", vertices\_with\_one\_edge)

else:

print("Немає висячих вершин")

def findIsolatedVertices(degree\_of\_vertex):

vertices\_without\_edges = [v for v, deg in degree\_of\_vertex.items() if deg == 0]

if vertices\_without\_edges:

print("Ізольовані вершини:", vertices\_without\_edges)

else:

print("Немає ізольованих вершин")

main()

*Програма для виконання пунктів 1, 2; напрямлений граф*

import turtle

import random

import math

class Vertex:

def \_\_init\_\_(self, number, pos\_x, pos\_y):

self.number = number

self.pos\_x = pos\_x

self.pos\_y = pos\_y

def main():

seed = 4416

n3 = 1

n4 = 6

N = 11

directed\_graph\_matrix = generateMatrix(n3, n4, N, seed)

undirected\_graph\_matrix = doUndir(directed\_graph\_matrix)

position = createPositions(undirected\_graph\_matrix)

degree\_of\_vertex = getDegreeOfVertex(undirected\_graph\_matrix)

checkForUniformity(degree\_of\_vertex)

findFinalVertices(degree\_of\_vertex)

findIsolatedVertices(degree\_of\_vertex)

drawVertices(position)

drawLines(position, undirected\_graph\_matrix)

drawCurvedLines(position, undirected\_graph\_matrix)

drawCircles(position, undirected\_graph\_matrix)

def generateMatrix(n3, n4, N, seed):

random.seed(seed)

adj\_matrix = [[random.random() \* 2.0 for \_ in range(N)] for \_ in range(N)]

k = 1.0 - n3 \* 0.01 - n4 \* 0.01 - 0.3

for i in range(N):

for j in range(N):

adj\_matrix[i][j] \*= k

adj\_matrix[i][j] = 0 if adj\_matrix[i][j] < 1.0 else 1

return adj\_matrix

def doUndir(adj\_matrix):

n = len(adj\_matrix)

undirected\_adj\_matrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

if adj\_matrix[i][j] == 1:

undirected\_adj\_matrix[i][j] = 1

undirected\_adj\_matrix[j][i] = 1

print("\nМатриця суміжності ненапрямленого графа:")

for row in undirected\_adj\_matrix:

print(row)

return undirected\_adj\_matrix

def createPositions(undirected\_graph\_matrix):

n = len(undirected\_graph\_matrix)

vertices = []

center\_x = 0

center\_y = 0

radius = 300

for i in range(1, n):

angle = i \* (2 \* math.pi / (n - 1))

x = center\_x + radius \* math.cos(angle)

y = center\_y + radius \* math.sin(angle)

vertices.append(Vertex(i, x, y))

central\_vertex = Vertex(n, center\_x, center\_y)

vertices.append(central\_vertex)

return vertices

def drawVertices(vertices):

turtle.speed(0)

turtle.penup()

radius = 20

for vertex in vertices:

x, y = vertex.pos\_x, vertex.pos\_y

turtle.goto(x, y - radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x, y)

turtle.write(vertex.number, align="center")

def drawLines(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

for i in range(n):

for j in range(i + 1, n):

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) != 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

angle = math.atan2(y2 - y1, x2 - x1)

x1 += math.cos(angle) \* radius

y1 += math.sin(angle) \* radius

x2 -= math.cos(angle) \* radius

y2 -= math.sin(angle) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.pendown()

turtle.goto(x2, y2)

def drawCurvedLines(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

for i in range(n):

for j in range(i + 1, n):

if i != j and undirected\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) == 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

angle = math.atan2(y2 - y1, x2 - x1)

x1 += math.cos(angle) \* radius

y1 += math.sin(angle) \* radius

x2 -= math.cos(angle) \* radius

y2 -= math.sin(angle) \* radius

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(math.degrees(angle))

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = math.pi \* 2

b = distance / 2 / math.cos(math.radians(degrees))

turtle.right(degrees)

turtle.forward(b)

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

def drawCircles(vertices, undirected\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 10

for i in range(n):

if undirected\_graph\_matrix[i][i] == 1:

x, y = vertices[i].pos\_x, vertices[i].pos\_y

turtle.goto(x + math.pi / 2.8 \* radius, y + math.pi / 2 \* radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.hideturtle()

turtle.done()

def getDegreeOfVertex(adjacency\_matrix):

num\_vertices = len(adjacency\_matrix)

edges = {}

for i in range(1, num\_vertices + 1):

edges[i] = sum(adjacency\_matrix[i - 1])

print("Степені вершин:", edges)

return edges

def checkForUniformity(degree\_of\_vertex):

edges\_counts = list(degree\_of\_vertex.values())

if len(set(edges\_counts)) == 1:

print("Граф однорідний. Степінь однорідності:", edges\_counts[0])

else:

print("Граф не є однорідним")

def findFinalVertices(degree\_of\_vertex):

vertices\_with\_one\_edge = [v for v, deg in degree\_of\_vertex.items() if deg == 1]

if vertices\_with\_one\_edge:

print("Висячі вершини:", vertices\_with\_one\_edge)

else:

print("Немає висячих вершин")

def findIsolatedVertices(degree\_of\_vertex):

vertices\_without\_edges = [v for v, deg in degree\_of\_vertex.items() if deg == 0]

if vertices\_without\_edges:

print("Ізольовані вершини:", vertices\_without\_edges)

else:

print("Немає ізольованих вершин")

main()

*Програма для виконання пунктів 3, 4*

import turtle

import random

import math

import time

class Vertex:

def \_\_init\_\_(self, number, pos\_x, pos\_y):

self.number = number

self.pos\_x = pos\_x

self.pos\_y = pos\_y

def main():

seed = 4416

n3 = 1

n4 = 6

N = 11

directed\_graph\_matrix = generateMatrix(n3, n4, N, seed)

position = createPositions(directed\_graph\_matrix)

countOutgoingEdges(directed\_graph\_matrix)

countIncomingEdges(directed\_graph\_matrix)

findPathsOfLengthTwo(directed\_graph\_matrix)

findPathsOfLengthThree(directed\_graph\_matrix)

reachability\_matrix = generateReachabilityMatrix(directed\_graph\_matrix)

strong\_connectivity\_matrix = generateStrongConnectivityMatrix(reachability\_matrix)

components\_of\_strong\_connectivity = getComponentsOfStrongConnectivity(strong\_connectivity\_matrix)

condensation\_matrix = generateAdjacencyMatrixOfCondensationGraph(directed\_graph\_matrix, components\_of\_strong\_connectivity)

positions\_of\_condensation\_graph = createPositions(condensation\_matrix)

drawVertices(position)

drawArrows1(position, directed\_graph\_matrix)

drawArrows2(position, directed\_graph\_matrix)

drawCurvedArrows1(position, directed\_graph\_matrix)

drawCurvedArrows2(position, directed\_graph\_matrix)

drawCirclesWithArrows(position, directed\_graph\_matrix)

turtle.hideturtle()

time.sleep(5)

turtle.reset()

drawVertices(positions\_of\_condensation\_graph)

drawArrows2(positions\_of\_condensation\_graph, condensation\_matrix)

turtle.hideturtle()

turtle.done()

def generateMatrix(n3, n4, N, seed):

random.seed(seed)

adj\_matrix = [[random.random() \* 2.0 for \_ in range(N)] for \_ in range(N)]

k = 1.0 - n3 \* 0.005 - n4 \* 0.005 - 0.27

for i in range(N):

for j in range(N):

adj\_matrix[i][j] \*= k

adj\_matrix[i][j] = 0 if adj\_matrix[i][j] < 1.0 else 1

print("Матриця суміжності напрямленого графа:")

for row in adj\_matrix:

print(row)

return adj\_matrix

def createPositions(matrix):

n = len(matrix)

vertices = []

center\_x = 0

center\_y = 0

radius = 300

for i in range(1, n):

angle = i \* (2 \* math.pi / (n-1))

x = center\_x + radius \* math.cos(angle)

y = center\_y + radius \* math.sin(angle)

vertex = Vertex(i, x, y)

vertices.append(vertex)

central\_vertex = Vertex(n, center\_x, center\_y)

vertices.append(central\_vertex)

return vertices

def drawVertices(vertices):

turtle.speed(0)

turtle.penup()

radius = 20

for vertex in vertices:

x, y = vertex.pos\_x, vertex.pos\_y

turtle.goto(x, y - radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x, y)

turtle.write(vertex.number, align="center")

def drawArrows1(vertices, directed\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

arrow\_size = 10

for i in range(n):

for j in range(i + 1, n):

if i != j and abs(vertices[i].number - vertices[j].number) != 5 and directed\_graph\_matrix[i][j] == 1:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

turtle.goto(x2, y2)

turtle.penup()

turtle.goto(x2, y2)

turtle.pendown()

turtle.right(150)

turtle.forward(arrow\_size)

turtle.penup()

turtle.goto(x2, y2)

turtle.left(300)

turtle.pendown()

turtle.forward(arrow\_size)

turtle.penup()

def drawArrows2(vertices, directed\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

arrow\_size = 10

for i in range(n):

for j in range(i):

if i != j and abs(vertices[i].number - vertices[j].number) != 5 and directed\_graph\_matrix[i][j] == 1 and directed\_graph\_matrix[j][i] != 1:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

turtle.goto(x2, y2)

turtle.penup()

turtle.goto(x2, y2)

turtle.pendown()

turtle.right(150)

turtle.forward(arrow\_size)

turtle.penup()

turtle.goto(x2, y2)

turtle.left(300)

turtle.pendown()

turtle.forward(arrow\_size)

turtle.penup()

def drawCurvedArrows1(vertices, directed\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

arrow\_size = 10

for i in range(n):

for j in range(i):

if directed\_graph\_matrix[i][j] == 1 and directed\_graph\_matrix[j][i] == 1 and abs(vertices[i].number - vertices[j].number) != 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = 10

b = distance / 2 / math.cos(math.radians(degrees))

turtle.left(degrees)

turtle.forward(b)

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

turtle.goto(x2, y2)

turtle.pendown()

turtle.right(150)

turtle.forward(arrow\_size)

turtle.penup()

turtle.goto(x2, y2)

turtle.left(300)

turtle.pendown()

turtle.forward(arrow\_size)

turtle.penup()

def drawCurvedArrows2(vertices, directed\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 20

arrow\_size = 10

for i in range(n):

for j in range(n):

if directed\_graph\_matrix[i][j] == 1 and abs(vertices[i].number - vertices[j].number) == 5:

x1, y1 = vertices[i].pos\_x, vertices[i].pos\_y

x2, y2 = vertices[j].pos\_x, vertices[j].pos\_y

x1 += math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) \* radius

y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) \* radius

angle = math.degrees(math.atan2(y2 - y1, x2 - x1))

turtle.penup()

turtle.goto(x1, y1)

turtle.setheading(angle)

turtle.pendown()

distance = turtle.distance(x2, y2)

degrees = 2 \* math.pi

b = distance / 2 / math.cos(math.radians(degrees))

turtle.right(degrees)

turtle.forward(b)

turtle.setheading(turtle.towards(x2, y2))

turtle.goto(x2, y2)

turtle.penup()

turtle.goto(x2, y2)

turtle.pendown()

turtle.right(150)

turtle.forward(arrow\_size)

turtle.penup()

turtle.goto(x2, y2)

turtle.left(300)

turtle.pendown()

turtle.forward(arrow\_size)

turtle.right(300)

turtle.penup()

def drawCirclesWithArrows(vertices, directed\_graph\_matrix):

turtle.speed(0)

turtle.penup()

n = len(vertices)

radius = 10

for i in range(n):

if directed\_graph\_matrix[i][i] == 1:

x, y = vertices[i].pos\_x, vertices[i].pos\_y

turtle.goto(x + math.pi / 1.8 \* radius, y + math.pi / 1.8 \* radius)

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.goto(x + math.pi / 1.6 \* radius - math.pi, y + math.pi / 1.9 \* radius - 1.4 \* math.pi)

turtle.pendown()

turtle.right(55)

turtle.backward(10)

turtle.penup()

turtle.goto(x + math.pi / 1.6 \* radius - math.pi, y + math.pi / 1.9 \* radius - 1.4 \* math.pi)

turtle.left(90)

turtle.pendown()

turtle.backward(10)

turtle.right(35)

turtle.penup()

def countOutgoingEdges(adjacency\_matrix):

num\_vertices = len(adjacency\_matrix)

outgoing\_edges = {}

for i in range(1, num\_vertices + 1):

outgoing\_edges[i] = sum(adjacency\_matrix[i - 1])

print("Напівстепені виходу вершин:", outgoing\_edges)

return outgoing\_edges

def countIncomingEdges(adjacency\_matrix):

num\_vertices = len(adjacency\_matrix)

transposed\_matrix = [[adjacency\_matrix[j][i] for j in range(num\_vertices)] for i in range(num\_vertices)]

incoming\_edges = {}

for i in range(1, num\_vertices + 1):

incoming\_edges[i] = sum(transposed\_matrix[i - 1])

print("Напівстепені заходу вершин:", incoming\_edges)

return incoming\_edges

def matrixMultiply(matrix1, matrix2):

result = [[0 for \_ in range(len(matrix2[0]))] for \_ in range(len(matrix1))]

for i in range(len(matrix1)):

for j in range(len(matrix2[0])):

for k in range(len(matrix2)):

result[i][j] += matrix1[i][k] \* matrix2[k][j]

return result

def matrixSum(matrix1, matrix2):

result = [[0] \* len(matrix1[0]) for \_ in range(len(matrix1))]

for i in range(len(matrix1)):

for j in range(len(matrix1[0])):

result[i][j] = matrix1[i][j] + matrix2[i][j]

return result

def transposeMatrix(matrix):

n = len(matrix)

transposedMatrix = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

transposedMatrix[j][i] = matrix[i][j]

return transposedMatrix

def multiplyMatricesElementByElement(matrix1, matrix2):

n = len(matrix1)

result = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(n):

result[i][j] = matrix1[i][j] \* matrix2[i][j]

return result

def findPathsOfLengthTwo(adjacency\_matrix):

square\_matrix = matrixMultiply(adjacency\_matrix, adjacency\_matrix)

numbers = len(square\_matrix)

print("Шляхи довжиною 2:")

for i in range(numbers):

for j in range(numbers):

if square\_matrix[i][j]:

for e in range(numbers):

if adjacency\_matrix[i][e] and adjacency\_matrix[e][j]:

print(f"{i + 1}-{e + 1}-{j + 1}")

def findPathsOfLengthThree(adjacency\_matrix):

cube\_matrix = matrixMultiply(matrixMultiply(adjacency\_matrix, adjacency\_matrix), adjacency\_matrix)

numbers = len(cube\_matrix)

print("Шляхи довжиною 3:")

for i in range(numbers):

for j in range(numbers):

if cube\_matrix[i][j]:

for k in range(numbers):

for m in range(numbers):

if adjacency\_matrix[i][k] and adjacency\_matrix[k][m] and adjacency\_matrix[m][j]:

print(f"{i + 1}-{k + 1}-{m + 1}-{j + 1}")

def generateReachabilityMatrix(directed\_graph\_matrix):

n = len(directed\_graph\_matrix)

previous\_degree = directed\_graph\_matrix

unit\_matrix = [[1 if i == j else 0 for j in range(n)] for i in range(n)]

reachability\_matrix = matrixSum(unit\_matrix, previous\_degree)

for \_ in range(n - 2):

previous\_degree = matrixMultiply(previous\_degree, directed\_graph\_matrix)

reachability\_matrix = matrixSum(reachability\_matrix, previous\_degree)

for i in range(n):

for j in range(n):

if reachability\_matrix[i][j] >= 1:

reachability\_matrix[i][j] = 1

else:

reachability\_matrix[i][j] = 0

print("Матриця досяжності:")

for row in reachability\_matrix:

print(row)

return reachability\_matrix

def generateStrongConnectivityMatrix(reachability\_matrix):

transposed\_reachability\_matrix = transposeMatrix(reachability\_matrix)

strong\_connectivity\_matrix = multiplyMatricesElementByElement(reachability\_matrix, transposed\_reachability\_matrix)

print("Матриця сильної зв'язності:")

for row in strong\_connectivity\_matrix:

print(row)

return strong\_connectivity\_matrix

def getComponentsOfStrongConnectivity(strong\_connectivity\_matrix):

def depthFirstSearch(vertex):

nonlocal component

visited[vertex] = True

component.append(vertex)

for neighbor in range(1, len(strong\_connectivity\_matrix) + 1):

if strong\_connectivity\_matrix[vertex - 1][neighbor - 1] == 1 and not visited[neighbor]:

depthFirstSearch(neighbor)

n = len(strong\_connectivity\_matrix)

visited = [False] \* (n + 1)

components = []

print("Компоненти сильної зв'язності:")

for vertex in range(1, n + 1):

if not visited[vertex]:

component = []

depthFirstSearch(vertex)

components.append(component)

print(f"{component}")

return components

def generateAdjacencyMatrixOfCondensationGraph(graph\_adjacency\_matrix, components):

num\_scc = len(components)

condensation\_matrix = [[0] \* num\_scc for \_ in range(num\_scc)]

for i in range(len(graph\_adjacency\_matrix)):

for j in range(len(graph\_adjacency\_matrix[0])):

if graph\_adjacency\_matrix[i][j] == 1:

scc\_i = next(index for index, scc in enumerate(components) if i + 1 in scc)

scc\_j = next(index for index, scc in enumerate(components) if j + 1 in scc)

if scc\_i != scc\_j:

condensation\_matrix[scc\_i][scc\_j] = 1

print("Матриця суміжності графа конденсації:")

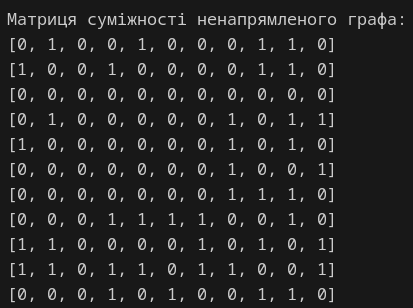
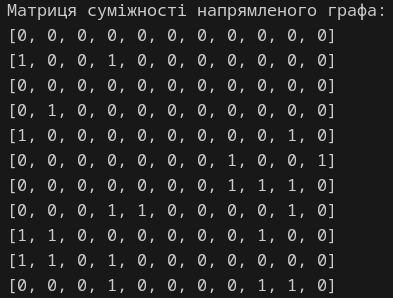
for row in condensation\_matrix:

print(row)

return condensation\_matrix

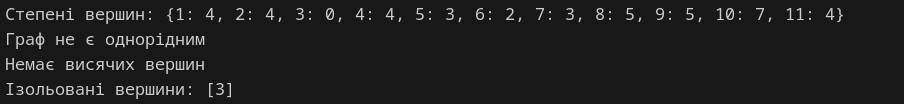
main()

**За п. 1: згенеровані матриці суміжності**

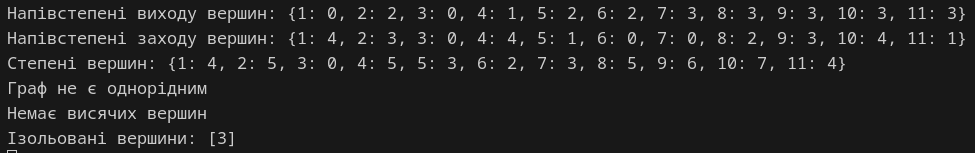
 

**За п. 2: перелік степенів, півстепенів, результат перевірки на однорідність, перелік висячих та ізольованих вершин**

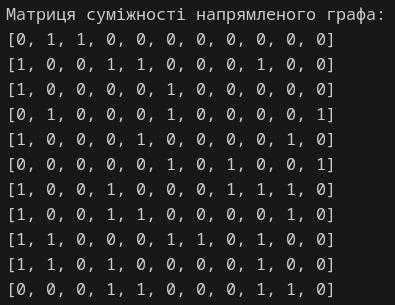
*Ненапрямлений граф*



*Напрямлений граф*



**За п. 3: матриця суміжності другого орграфа**



**За п. 4: перелiки пiвстепенiв, шляхiв, матрицi досяжностi та сильної зв’язностi, перелiк компонент сильної зв’язностi, граф конденсацiї**



*Шляхи довжиною 2:*

1-2-1

1-3-1

1-2-4

1-2-5

1-3-6

1-2-9

2-5-1

2-9-1

2-1-2

2-4-2

2-9-2

2-1-3

2-5-5

2-4-6

2-9-6

2-9-7

2-9-9

2-5-10

2-4-11

3-1-2

3-1-3

3-6-6

3-6-8

3-6-11

4-2-1

4-2-4

4-11-4

4-2-5

4-11-5

4-6-6

4-6-8

4-2-9

4-11-9

4-11-10

4-6-11

5-5-1

5-10-1

5-1-2

5-10-2

5-1-3

5-10-4

5-5-5

5-10-9

5-5-10

6-8-1

6-8-4

6-11-4

6-8-5

6-11-5

6-6-6

6-6-8

6-11-9

6-8-10

6-11-10

6-6-11

7-8-1

7-9-1

7-10-1

7-1-2

7-4-2

7-9-2

7-10-2

7-1-3

7-8-4

7-10-4

7-8-5

7-4-6

7-9-6

7-9-7

7-9-9

7-10-9

7-8-10

7-4-11

8-5-1

8-10-1

8-1-2

8-4-2

8-10-2

8-1-3

8-10-4

8-5-5

8-4-6

8-10-9

8-5-10

8-4-11

9-2-1

9-7-1

9-9-1

9-1-2

9-9-2

9-1-3

9-2-4

9-7-4

9-2-5

9-6-6

9-9-6

9-9-7

9-6-8

9-7-8

9-2-9

9-7-9

9-9-9

9-7-10

9-6-11

10-2-1

10-9-1

10-1-2

10-4-2

10-9-2

10-1-3

10-2-4

10-2-5

10-4-6

10-9-6

10-9-7

10-2-9

10-9-9

10-4-11

11-5-1

11-9-1

11-10-1

11-4-2

11-9-2

11-10-2

11-10-4

11-5-5

11-4-6

11-9-6

11-9-7

11-9-9

11-10-9

11-5-10

11-4-11

*Шляхи довжиною 3:*

1-2-5-1

1-2-9-1

1-2-1-2

1-2-4-2

1-2-9-2

1-3-1-2

1-2-1-3

1-3-1-3

1-2-5-5

1-2-4-6

1-2-9-6

1-3-6-6

1-2-9-7

1-3-6-8

1-2-9-9

1-2-5-10

1-2-4-11

1-3-6-11

2-1-2-1

2-1-3-1

2-4-2-1

2-5-5-1

2-5-10-1

2-9-2-1

2-9-7-1

2-9-9-1

2-5-1-2

2-5-10-2

2-9-1-2

2-9-9-2

2-5-1-3

2-9-1-3

2-1-2-4

2-4-2-4

2-4-11-4

2-5-10-4

2-9-2-4

2-9-7-4

2-1-2-5

2-4-2-5

2-4-11-5

2-5-5-5

2-9-2-5

2-1-3-6

2-4-6-6

2-9-6-6

2-9-9-6

2-9-9-7

2-4-6-8

2-9-6-8

2-9-7-8

2-1-2-9

2-4-2-9

2-4-11-9

2-5-10-9

2-9-2-9

2-9-7-9

2-9-9-9

2-4-11-10

2-5-5-10

2-9-7-10

2-4-6-11

2-9-6-11

3-1-2-1

3-1-3-1

3-6-8-1

3-1-2-4

3-6-8-4

3-6-11-4

3-1-2-5

3-6-8-5

3-6-11-5

3-1-3-6

3-6-6-6

3-6-6-8

3-1-2-9

3-6-11-9

3-6-8-10

3-6-11-10

3-6-6-11

4-2-5-1

4-2-9-1

4-6-8-1

4-11-5-1

4-11-9-1

4-11-10-1

4-2-1-2

4-2-4-2

4-2-9-2

4-11-4-2

4-11-9-2

4-11-10-2

4-2-1-3

4-6-8-4

4-6-11-4

4-11-10-4

4-2-5-5

4-6-8-5

4-6-11-5

4-11-5-5

4-2-4-6

4-2-9-6

4-6-6-6

4-11-4-6

4-11-9-6

4-2-9-7

4-11-9-7

4-6-6-8

4-2-9-9

4-6-11-9

4-11-9-9

4-11-10-9

4-2-5-10

4-6-8-10

4-6-11-10

4-11-5-10

4-2-4-11

4-6-6-11

4-11-4-11

5-1-2-1

5-1-3-1

5-5-5-1

5-5-10-1

5-10-2-1

5-10-9-1

5-5-1-2

5-5-10-2

5-10-1-2

5-10-4-2

5-10-9-2

5-5-1-3

5-10-1-3

5-1-2-4

5-5-10-4

5-10-2-4

5-1-2-5

5-5-5-5

5-10-2-5

5-1-3-6

5-10-4-6

5-10-9-6

5-10-9-7

5-1-2-9

5-5-10-9

5-10-2-9

5-10-9-9

5-5-5-10

5-10-4-11

6-6-8-1

6-8-5-1

6-8-10-1

6-11-5-1

6-11-9-1

6-11-10-1

6-8-1-2

6-8-4-2

6-8-10-2

6-11-4-2

6-11-9-2

6-11-10-2

6-8-1-3

6-6-8-4

6-6-11-4

6-8-10-4

6-11-10-4

6-6-8-5

6-6-11-5

6-8-5-5

6-11-5-5

6-6-6-6

6-8-4-6

6-11-4-6

6-11-9-6

6-11-9-7

6-6-6-8

6-6-11-9

6-8-10-9

6-11-9-9

6-11-10-9

6-6-8-10

6-6-11-10

6-8-5-10

6-11-5-10

6-6-6-11

6-8-4-11

6-11-4-11

7-1-2-1

7-1-3-1

7-4-2-1

7-8-5-1

7-8-10-1

7-9-2-1

7-9-7-1

7-9-9-1

7-10-2-1

7-10-9-1

7-8-1-2

7-8-4-2

7-8-10-2

7-9-1-2

7-9-9-2

7-10-1-2

7-10-4-2

7-10-9-2

7-8-1-3

7-9-1-3

7-10-1-3

7-1-2-4

7-4-2-4

7-4-11-4

7-8-10-4

7-9-2-4

7-9-7-4

7-10-2-4

7-1-2-5

7-4-2-5

7-4-11-5

7-8-5-5

7-9-2-5

7-10-2-5

7-1-3-6

7-4-6-6

7-8-4-6

7-9-6-6

7-9-9-6

7-10-4-6

7-10-9-6

7-9-9-7

7-10-9-7

7-4-6-8

7-9-6-8

7-9-7-8

7-1-2-9

7-4-2-9

7-4-11-9

7-8-10-9

7-9-2-9

7-9-7-9

7-9-9-9

7-10-2-9

7-10-9-9

7-4-11-10

7-8-5-10

7-9-7-10

7-4-6-11

7-8-4-11

7-9-6-11

7-10-4-11

8-1-2-1

8-1-3-1

8-4-2-1

8-5-5-1

8-5-10-1

8-10-2-1

8-10-9-1

8-5-1-2

8-5-10-2

8-10-1-2

8-10-4-2

8-10-9-2

8-5-1-3

8-10-1-3

8-1-2-4

8-4-2-4

8-4-11-4

8-5-10-4

8-10-2-4

8-1-2-5

8-4-2-5

8-4-11-5

8-5-5-5

8-10-2-5

8-1-3-6

8-4-6-6

8-10-4-6

8-10-9-6

8-10-9-7

8-4-6-8

8-1-2-9

8-4-2-9

8-4-11-9

8-5-10-9

8-10-2-9

8-10-9-9

8-4-11-10

8-5-5-10

8-4-6-11

8-10-4-11

9-1-2-1

9-1-3-1

9-2-5-1

9-2-9-1

9-6-8-1

9-7-8-1

9-7-9-1

9-7-10-1

9-9-2-1

9-9-7-1

9-9-9-1

9-2-1-2

9-2-4-2

9-2-9-2

9-7-1-2

9-7-4-2

9-7-9-2

9-7-10-2

9-9-1-2

9-9-9-2

9-2-1-3

9-7-1-3

9-9-1-3

9-1-2-4

9-6-8-4

9-6-11-4

9-7-8-4

9-7-10-4

9-9-2-4

9-9-7-4

9-1-2-5

9-2-5-5

9-6-8-5

9-6-11-5

9-7-8-5

9-9-2-5

9-1-3-6

9-2-4-6

9-2-9-6

9-6-6-6

9-7-4-6

9-7-9-6

9-9-6-6

9-9-9-6

9-2-9-7

9-7-9-7

9-9-9-7

9-6-6-8

9-9-6-8

9-9-7-8

9-1-2-9

9-2-9-9

9-6-11-9

9-7-9-9

9-7-10-9

9-9-2-9

9-9-7-9

9-9-9-9

9-2-5-10

9-6-8-10

9-6-11-10

9-7-8-10

9-9-7-10

9-2-4-11

9-6-6-11

9-7-4-11

9-9-6-11

10-1-2-1

10-1-3-1

10-2-5-1

10-2-9-1

10-4-2-1

10-9-2-1

10-9-7-1

10-9-9-1

10-2-1-2

10-2-4-2

10-2-9-2

10-9-1-2

10-9-9-2

10-2-1-3

10-9-1-3

10-1-2-4

10-4-2-4

10-4-11-4

10-9-2-4

10-9-7-4

10-1-2-5

10-2-5-5

10-4-2-5

10-4-11-5

10-9-2-5

10-1-3-6

10-2-4-6

10-2-9-6

10-4-6-6

10-9-6-6

10-9-9-6

10-2-9-7

10-9-9-7

10-4-6-8

10-9-6-8

10-9-7-8

10-1-2-9

10-2-9-9

10-4-2-9

10-4-11-9

10-9-2-9

10-9-7-9

10-9-9-9

10-2-5-10

10-4-11-10

10-9-7-10

10-2-4-11

10-4-6-11

10-9-6-11

11-4-2-1

11-5-5-1

11-5-10-1

11-9-2-1

11-9-7-1

11-9-9-1

11-10-2-1

11-10-9-1

11-5-1-2

11-5-10-2

11-9-1-2

11-9-9-2

11-10-1-2

11-10-4-2

11-10-9-2

11-5-1-3

11-9-1-3

11-10-1-3

11-4-2-4

11-4-11-4

11-5-10-4

11-9-2-4

11-9-7-4

11-10-2-4

11-4-2-5

11-4-11-5

11-5-5-5

11-9-2-5

11-10-2-5

11-4-6-6

11-9-6-6

11-9-9-6

11-10-4-6

11-10-9-6

11-9-9-7

11-10-9-7

11-4-6-8

11-9-6-8

11-9-7-8

11-4-2-9

11-4-11-9

11-5-10-9

11-9-2-9

11-9-7-9

11-9-9-9

11-10-2-9

11-10-9-9

11-4-11-10

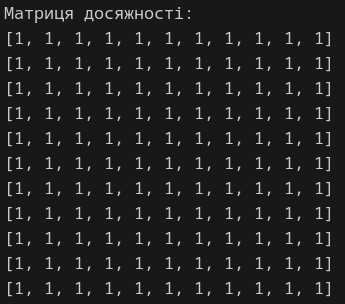
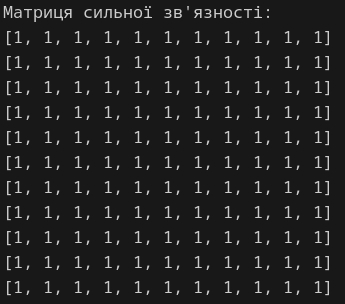
11-5-5-10

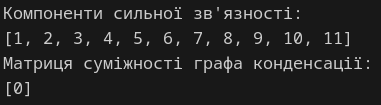
11-9-7-10

11-4-6-11

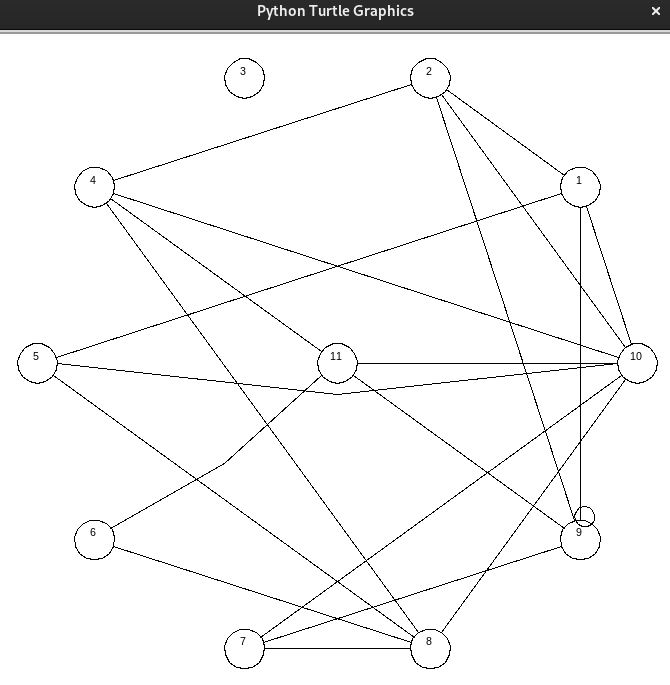
11-9-6-11

11-10-4-11

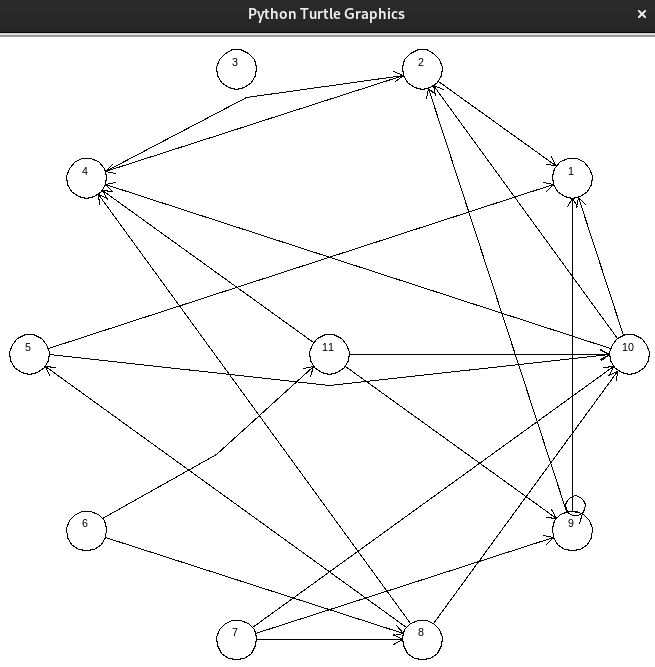
 



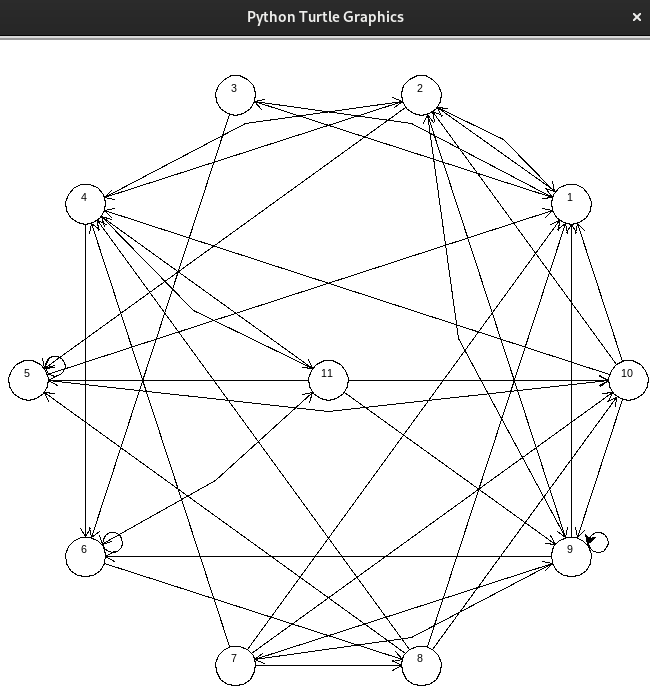
**Графічне представлення графів**



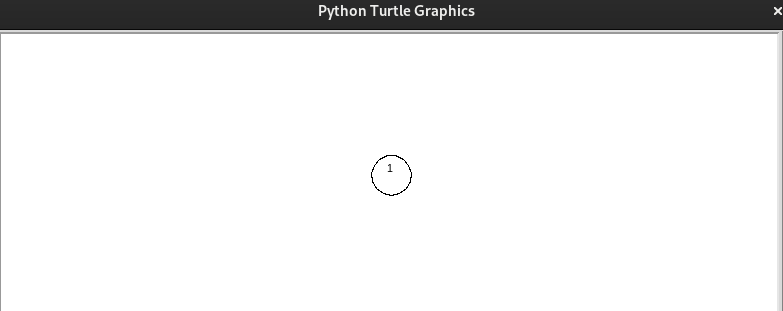
*Пункт 1, 2; ненапрямлений граф*



*Пункт 1, 2; напрямлений граф*



*Пункт 3, 4; напрямлений граф*



*Граф конденсації*

**Висновок**: під час виконання цієї лабораторної роботи я дослідив основні характеристики графів, такі як: степені, напівстепені вершин; однорідність графа; ізольовані та висячі вершини; маршрути від вершини до вершини; матриця досяжності; матриця сильної зв’язності; компоненти сильної зв’язності; граф конденсації. Також я навчився працювати з матрицями, виконувати над ними такі дії, як: множення, поелементне множення, додавання, транзитивне замикання. Після завершення виконання завдань цієї лабораторної роботи я здобув навички роботи з графом, навчився визначати його характеристики на конкретних прикладах.