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-44
Мундурс Нікіта Юрійович
номер у списку групи: 16

Перевірила:

Сергієнко М. А.

Київ 2025

Завдання

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

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

   Матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:
   1) встановлюється параметр (seed) генератора випадкових чисел, рівне 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 в 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 в графічне вікно, консоль або файл.

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

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

Граф конденсації вивести в графічне вікно.

При проєктуванні програм ***слід 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

k_1 = 0,63

k_2 = 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

```
Матриця суміжності ненапрямленого графа:
[0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0]
[1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1]
[1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1]
[1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0]
```

```
Матриця суміжності напрямленого графа:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0]
```

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

Ненапрямлений граф

```
Степені вершин: {1: 4, 2: 4, 3: 0, 4: 4, 5: 3, 6: 2, 7: 3, 8: 5, 9: 5, 10: 7, 11: 4}
Граф не є однорідним
Немає висячих вершин
Ізольовані вершини: [3]
```

Напрямлений граф

```
Напівстепені виходу вершин: {1: 0, 2: 2, 3: 0, 4: 1, 5: 2, 6: 2, 7: 3, 8: 3, 9: 3, 10: 3, 11: 3}
Напівстепені заходу вершин: {1: 4, 2: 3, 3: 0, 4: 4, 5: 1, 6: 0, 7: 0, 8: 2, 9: 3, 10: 4, 11: 1}
Степені вершин: {1: 4, 2: 5, 3: 0, 4: 5, 5: 3, 6: 2, 7: 3, 8: 5, 9: 6, 10: 7, 11: 4}
Граф не є однорідним
Немає висячих вершин
Ізольовані вершини: [3]
```

За п. 3: матриця суміжності другого орграфа

```
Матриця суміжності напрямленого графа:
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1]
[1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0]
[1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0]
```

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

```
Напівстепені виходу вершин: {1: 2, 2: 4, 3: 2, 4: 3, 5: 3, 6: 3, 7: 5, 8: 4, 9: 5, 10: 4, 11: 4}
Напівстепені заходу вершин: {1: 7, 2: 4, 3: 1, 4: 5, 5: 4, 6: 4, 7: 1, 8: 2, 9: 5, 10: 4, 11: 2}
```

Шляхи довжиною 2:

1-2-1	2-9-1	2-9-6
1-3-1	2-1-2	2-9-7
1-2-4	2-4-2	2-9-9
1-2-5	2-9-2	2-5-10
1-3-6	2-1-3	2-4-11
1-2-9	2-5-5	3-1-2
2-5-1	2-4-6	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

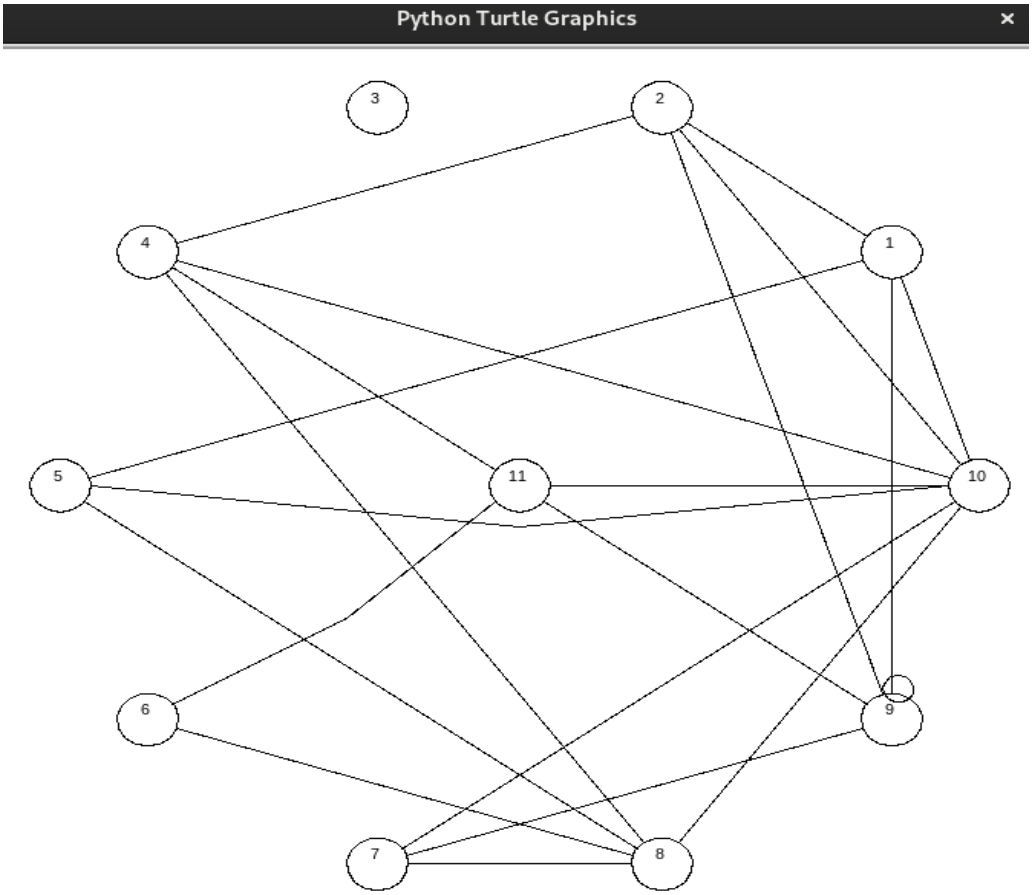
11-4-11-4
11-5-10-4
11-9-2-4
11-9-7-4
11-10-2-4
11-4-2-5
11-4-11-5
11-5-5-5
11-9-2-5
11-10-2-5
11-4-6-6
11-9-6-6
11-9-9-6
11-10-4-6
11-10-9-6
11-9-9-7
11-10-9-7
11-4-6-8
11-9-6-8
11-9-7-8
11-4-2-9
11-4-11-9
11-5-10-9
11-9-2-9
11-9-7-9
11-9-9-9
11-10-2-9
11-10-9-9
11-4-11-10
11-5-5-10
11-9-7-10
11-4-6-11
11-9-6-11
11-10-4-11

```
Матриця досяжності:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

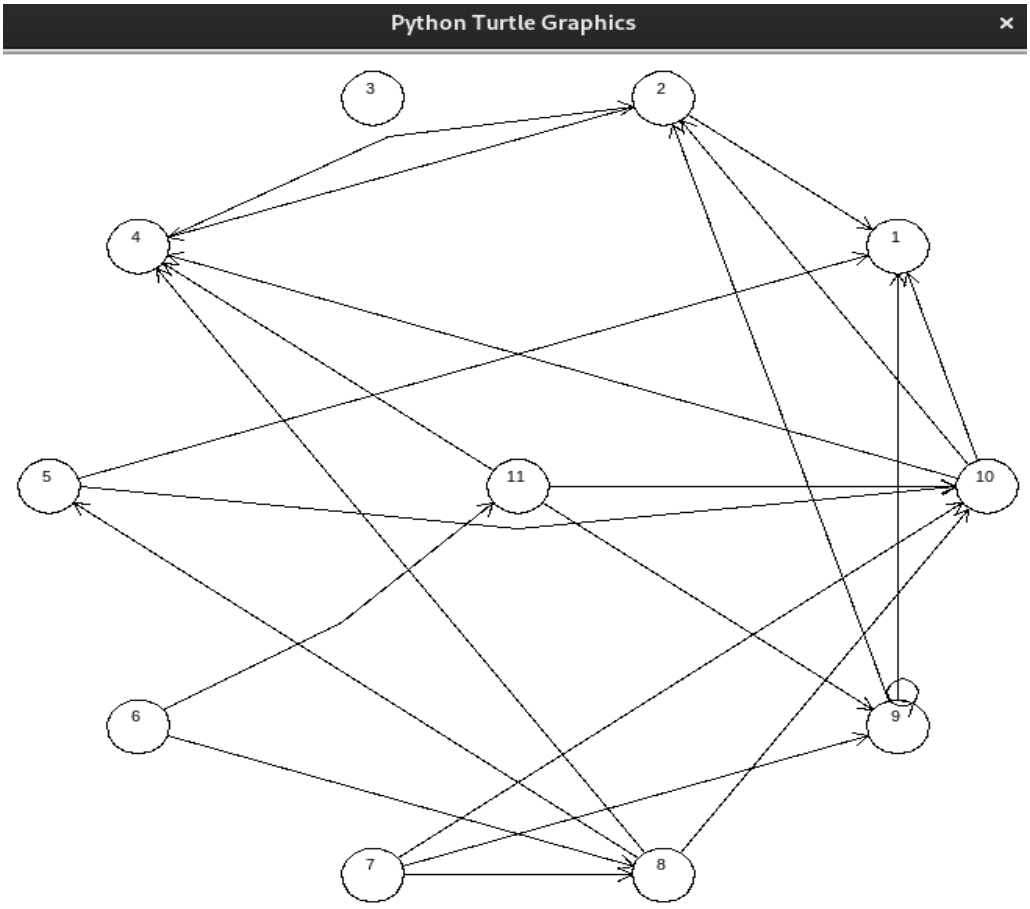
```
Матриця сильної зв'язності:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
Компоненти сильної зв'язності:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Матриця суміжності графа конденсації:
[0]
```

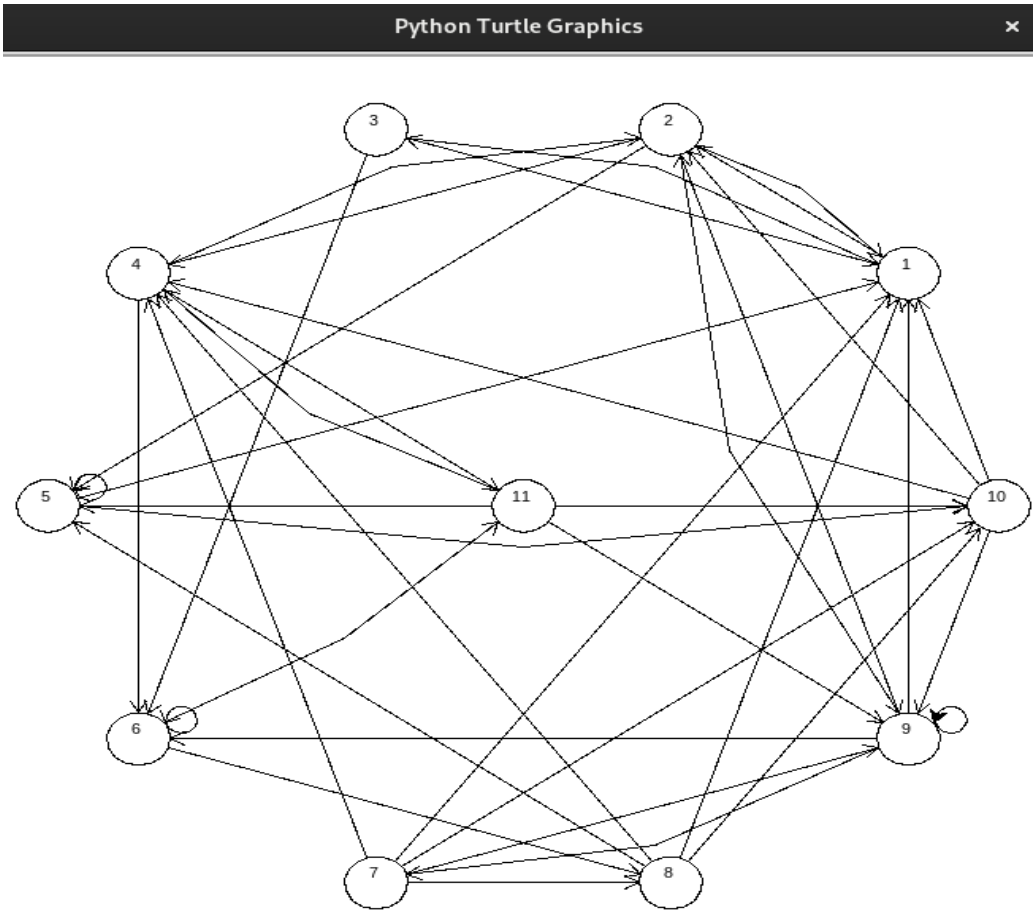
Графічне представлення графів



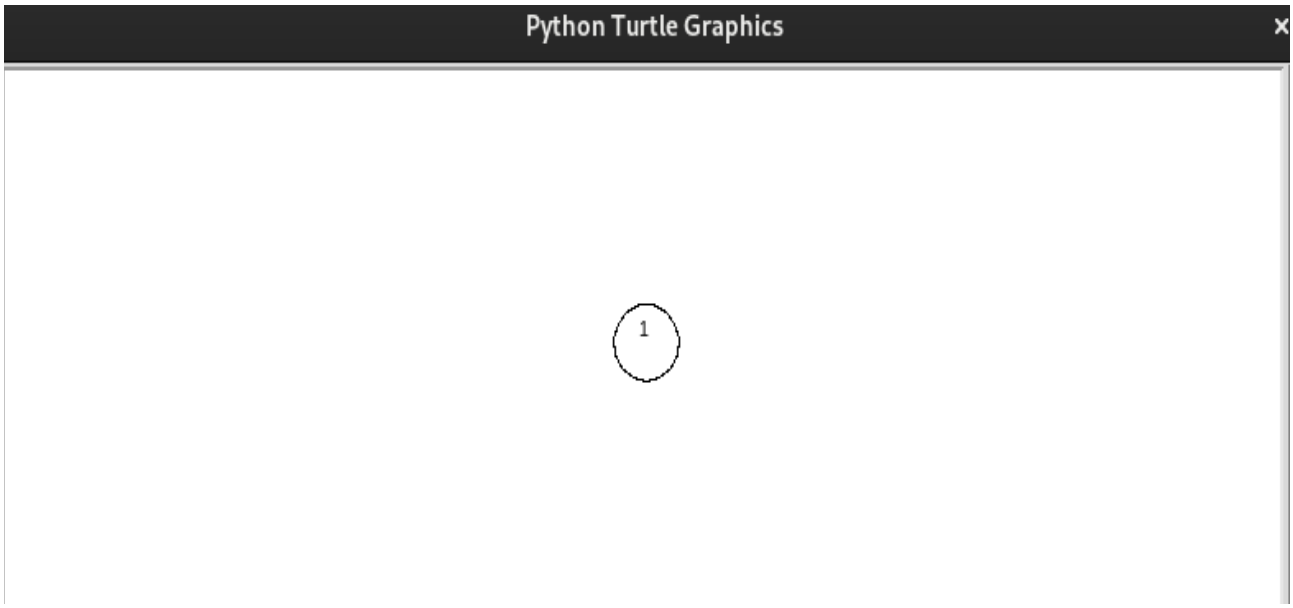
Пункт 1, 2; ненапрямлений граф



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.