**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 №3**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірив:

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

Київ 2025

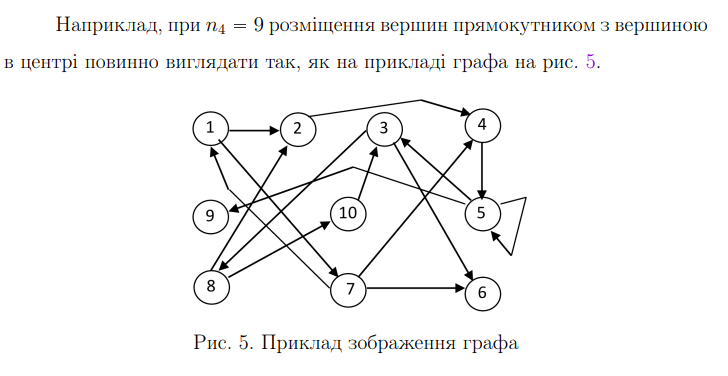
**Завдання**

1. Представити у програмi напрямлений i ненапрямлений графи з заданими параметрами:
2. кількість вершин *n;*
3. розміщення вершин;
4. матриця суміжності A.
5. Створити програму для формування зображення напрямленого i ненапрямленого графiв у графiчному вiкнi.

Згаданi вище параметри графа задаються на основi чотиризначного номера варiанту *n1n2n3n4*, де *n1n2* це десятковi цифри номера групи, а *n3n4* — десятковi цифри номера варiанту, який був у студента для двох попереднiх робiт (див. таблицю з поточними оцiнками з АСД, надану викладачем на початку поточного семестру).

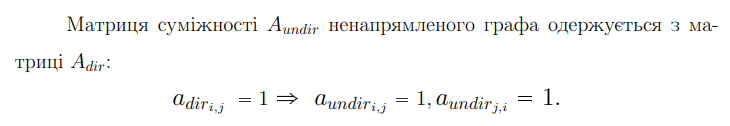
***Кiлькiсть вершин*** *n* дорiвнює 10+n3.

***Розмiщення вершин***:

1. колом при *n4* = 0, 1;
2. квадратом (прямокутником) при *n4* = 2, 3;
3. трикутником при *n4* = 4, 5;
4. колом з вершиною в центрі при *n4* = 6, 7;
5. квадратом (прямокутником) з вершиною в центрі при *n4* = 8, 9.

Матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1. встановлюється зерно 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* \* 0.02 - *n4* \* 0.005 – 0.25;
4. кожен елемент матриці множиться на коефіцієнт *k*;
5. елементи матриці округлюються: 0 – якщо елемент менший за 1.0, 1 – якщо елемент більший або дорівнює 1.0.



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

1. мова програмування обирається студентом самостійно;
2. графічне зображення графа має формуватися на основі графічних примітивів з графічної бібліотеки (таких як еліпс, пряма, дуга, текст тощо);
3. використання готових бібліотек для роботи з графами не дозволяється;
4. вивід графа має бути реалізований універсальним чином: вершини і ребра мають виводитися в циклі, а не окремими командами для кожного графічного елемента
5. типи та структури даних для внутрішнього представлення графа в програмі слід вибрати самостійно;
6. матриці суміжності графів можна виводити в графічне вікно або консоль – на розсуд студента;
7. матриці суміжності мають виводитися як матриці: у квадратному вигляді, з 1 та 0.

**Варіант:** 4416

**Кількість вершин:** 11

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

**Обрана мова програмування**: Python

**Тексти програм**

*Програма для графічного представлення ненапрямленого графа*

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)

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 \* 0.02 - n4 \* 0.005 - 0.25

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):

vertices = []

center\_x = 0

center\_y = 0

radius = 300

n = len(undirected\_graph\_matrix)

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 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): # верхній трикутник матриці

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

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

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): # верхній трикутник матриці

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

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 = 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): # головна діагональ

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

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

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

turtle.pendown()

turtle.circle(radius)

turtle.penup()

turtle.hideturtle()

turtle.done()

main()

*Програма для графічного представлення напрямленого графа*

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)

position = createPositions(directed\_graph\_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)

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 \* 0.02 - n4 \* 0.005 - 0.25

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(directed\_graph\_matrix):

vertices = []

center\_x = 0

center\_y = 0

radius = 300

n = len(directed\_graph\_matrix)

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): # верхній трикутник матриці

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): # нижній трикутник матриці

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): # нижній трикутник матриці

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): # повна матриця

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): # головна діагональ

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)

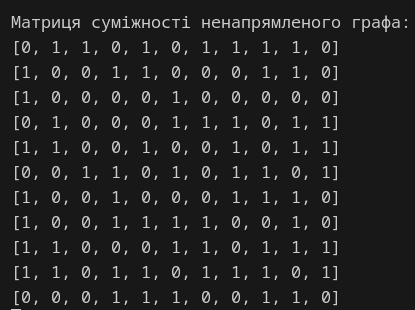
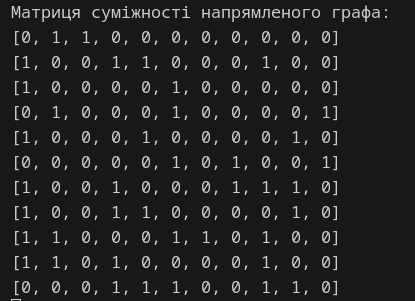
turtle.penup()

turtle.hideturtle()

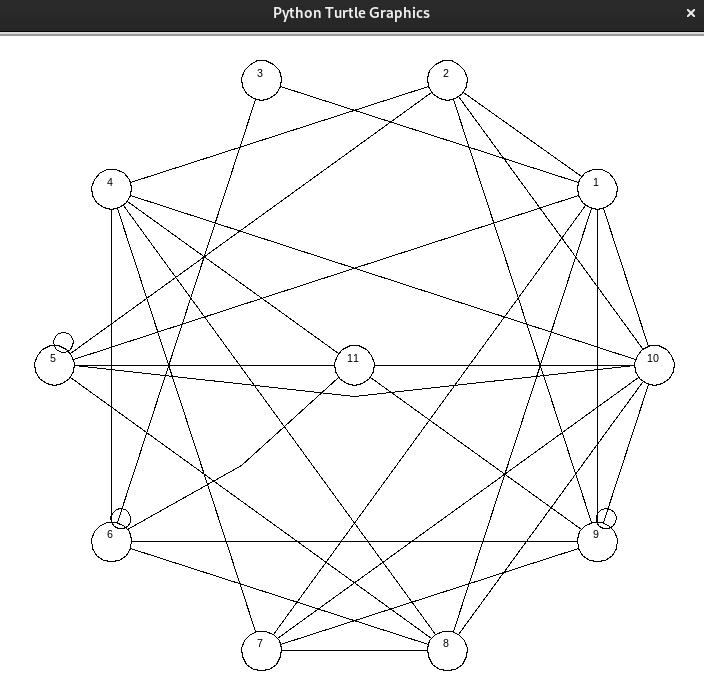
turtle.done()

main()

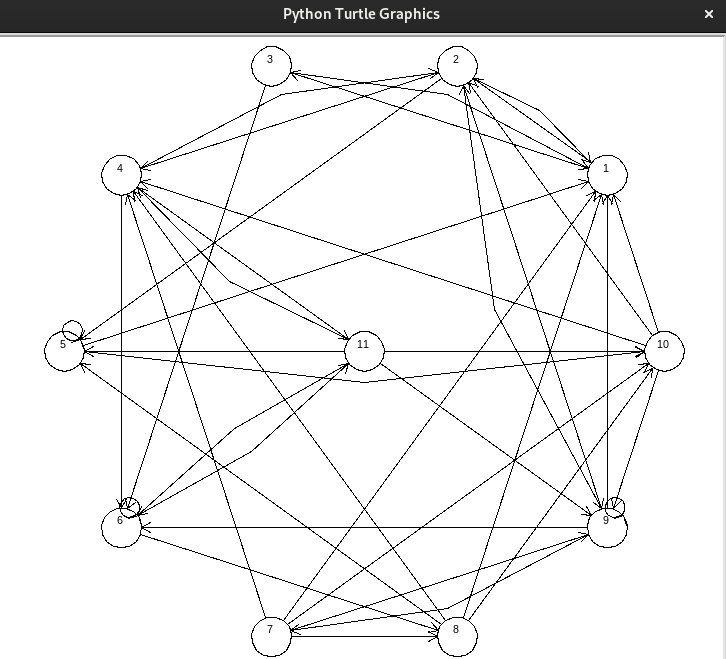
**Матриці суміжності**

**Графічне представлення графів**



*Ненапрямлений граф*



*Напрямлений граф*

**Висновок.** У ході виконання лабораторної роботи я навчився представляти графи за допомогою матриці суміжності та графічно виводити їх на екран. Я попрактикувався у використанні мови Python для побудови як напрямлених, так і ненапрямлених графів. Також я закріпив навички роботи з генераторами випадкових чисел та графічними примітивами. Під час реалізації було використано циклічні структури для універсального відображення вершин і ребер. Я отримав досвід створення алгоритмів для обробки та візуалізації графових структур. Важливою частиною роботи було правильне розміщення вершин відповідно до заданої геометричної форми. Я зрозумів принципи побудови матриці суміжності з урахуванням заданого коефіцієнта. Також я покращив розуміння способів представлення зв’язків між об’єктами у вигляді графа. У процесі роботи я навчився поєднувати математичну логіку з візуальним відображенням даних. Загалом, ця робота сприяла глибшому засвоєнню понять структур даних та графових алгоритмів.