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:

студент групи ІМ-44
Мундурс Нікіта Юрійович
номер у списку групи: 16

Перевірив:

Сергієнко М. А.

Київ 2025

Завдання

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:
    1) кількість вершин *n;*
    2) розміщення вершин;
    3) матриця суміжності A.
2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Згадані вище параметри графа задаються на основі чотиризначного номера варіанту $n_1n_2n_3n_4$, де n_1n_2 це десяткові цифри номера групи, а n_3n_4 — десяткові цифри номера варіанту, який був у студента для двох попередніх робіт (див. таблицю з поточними оцінками з АСД, надану викладачем на початку поточного семестру).

Кількість вершин *n* дорівнює $10+n_3$.

Розміщення вершин:

1) колом при $n_4 = 0, 1$;
2) квадратом (прямокутником) при $n_4 = 2, 3$;
3) трикутником при $n_4 = 4, 5$;
4) колом з вершиною в центрі при $n_4 = 6, 7$;
5) квадратом (прямокутником) з вершиною в центрі при $n_4 = 8, 9$.

Наприклад, при $n_4 = 9$ розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа на рис. 5.

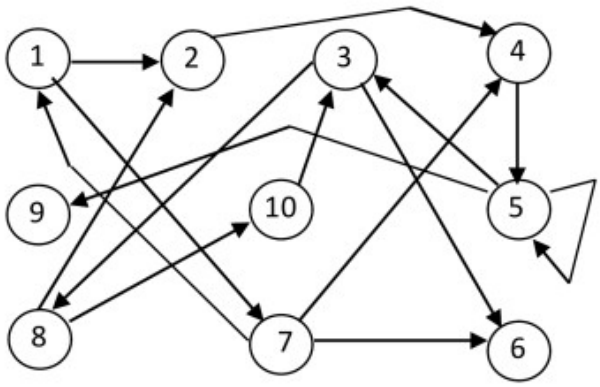


Рис. 5. Приклад зображення графа

Матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

1) встановлюється зерно 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 $k = 1.0 - n_3 * 0.02 - n_4 * 0.005 - 0.25$;
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.

Матриця суміжності A_{undir} ненапрямленого графа одержується з матриці A_{dir}:

$$a_{dir_{i,j}} = 1 \Rightarrow a_{undir_{i,j}} = 1, a_{undir_{j,i}} = 1.$$

При проєктуванні програм ***слід 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()
```

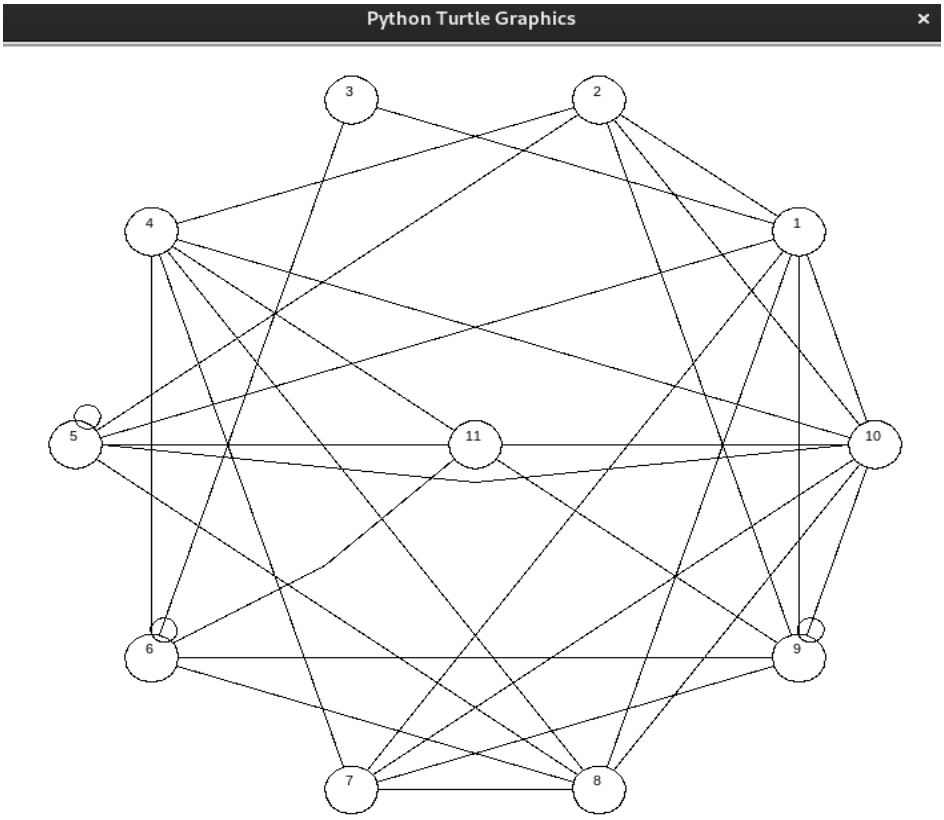
```
  turtle.hideturtle()
  turtle.done()

main()
```

Матриці суміжності

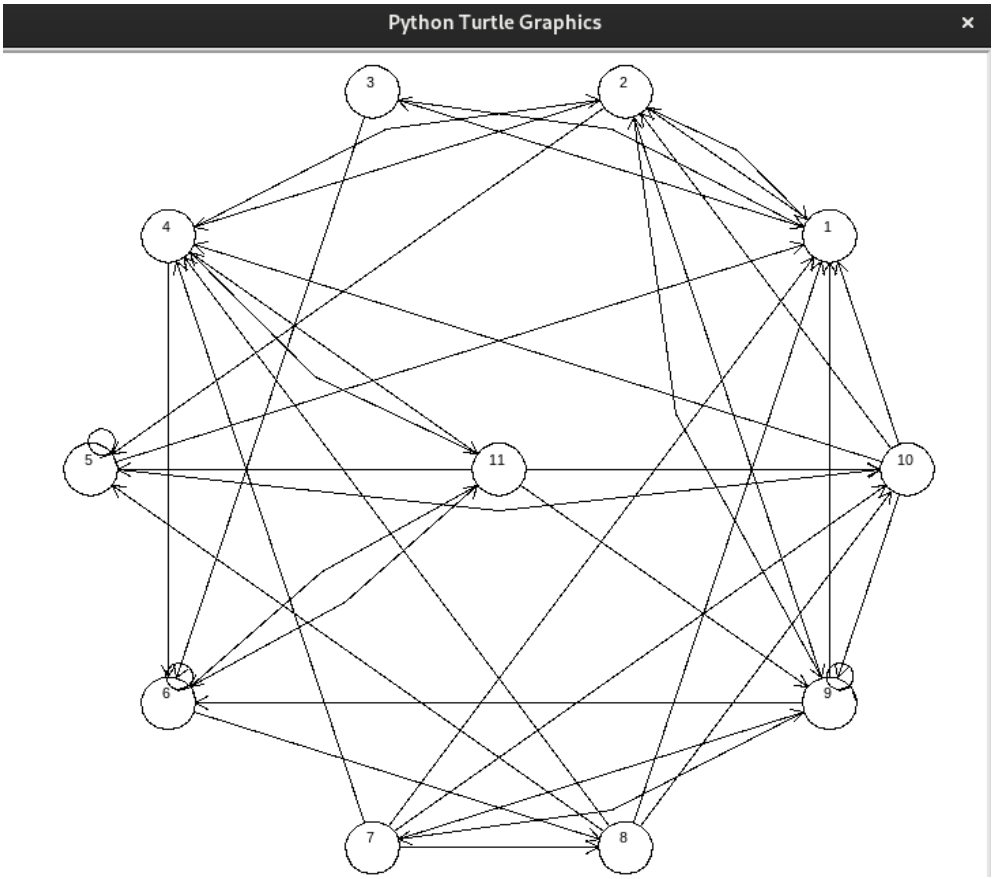
```
Матриця суміжності ненапрямленого графа:
[0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0]
[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1]
[1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1]
[0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1]
[1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 1,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

```
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, 1, 0, 0, 1, 1, 0]
```

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.