Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №5

з дисципліни
«Алгоритми і структури даних»

Виконав:

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

Перевірив:

Сергієнко М. А.

Київ 2025

Завдання

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

   Відмінність: коефіцієнт $k = 1.0 – n_3 * 0.01 - n_4 * 0.005 – 0.15$

   Отже, 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 $k = 1.0 – n_3 * 0.01 - n_4 * 0.005 – 0.15$, 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. Створити програму, яка виконує обхід напрямленого графа вшир (BFS) та вглиб (DFS):
   1) обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;
   2) при обході враховувати порядок нумерації;
   3) у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3. Під час обходу графа побудувати дерево обходу. У програмі дерево обходу виводити покроково в процесі виконання обходу графа. Це можна виконати одним із двох способів:
   1) або виділяти іншим кольором ребра графа;
   2) або будувати дерево обходу поряд із графом.

4. Зміну статусів вершин у процесі обходу продемонструвати зміною кольорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу в графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжувати обхід з невідвіданої вершини з найменшим номером, яка має щонайменше одну вихідну дугу.

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 обов’язково зображувати в графічному вікні;

4) типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;

Варіант: 4416

k **=** 0,81

Кількість вершин: 11

Розміщення вершин: колом з вершиною в центрі

Обрана мова програмування: Python

Тексти програм

Програма для реалізації обходу графа в глибину (DFS)

```
import turtle
import random
import math
import time



class Stack:
  def __init__(self):
    self.items = []



  def isEmpty(self):
    return self.items == []



  def push(self, item):
    self.items.append(item)



  def pop(self):
```

```
    return self.items.pop()


  def peek(self):
    return self.items[len(self.items) - 1]


  def size(self):
    return len(self.items)


  def is_empty(self):
    return len(self.items) == 0


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)
```

```
  drawArrows(position, directed_graph_matrix)
  dfs(directed_graph_matrix, position)
  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 * 0.01 - n4 * 0.005 - 0.15
  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):
  n = len(directed_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)
    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 drawArrows(vertices, directed_graph_matrix):
  turtle.speed(0)
  turtle.penup()
  n = len(vertices)
  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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

        turtle.goto(x1, y1)
        turtle.setheading(angle)
        turtle.pendown()
        turtle.goto(x2, y2)
        turtle.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()

  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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

        turtle.goto(x1, y1)
        turtle.setheading(angle)
        turtle.pendown()
        turtle.goto(x2, y2)
        turtle.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()

    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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
```

```
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

        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.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()

  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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

        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.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()

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 * 10, y + math.pi / 1.8 * 10)
        turtle.pendown()
        turtle.circle(10)
        turtle.penup()
        turtle.goto(x + math.pi / 1.6 * 10 - math.pi, y + math.pi / 1.9 * 10 - 1.4 * math.pi)
        turtle.pendown()
        turtle.right(55)
        turtle.backward(10)
        turtle.penup()
        turtle.goto(x + math.pi / 1.6 * 10 - math.pi, y + math.pi / 1.9 * 10 - 1.4 * math.pi)
        turtle.left(90)
        turtle.pendown()
        turtle.backward(10)
        turtle.right(35)
        turtle.penup()


def drawColoredArrows(vertices, i, j):
  turtle.speed(0)
  turtle.penup()
  turtle.pensize(2)
  turtle.color("red")
  arrow_size = 10

  if 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)) * 20
```

```
    y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
    x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
    y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

    turtle.goto(x1, y1)
    turtle.setheading(angle)
    turtle.pendown()
    turtle.goto(x2, y2)
    turtle.right(150)
    turtle.forward(arrow_size)
    turtle.penup()
    turtle.goto(x2, y2)
    turtle.left(300)
    turtle.pendown()
    turtle.forward(arrow_size)
    turtle.penup()

if 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)) * 20
    y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
    x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
    y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

    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.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 getStartVertex(adj_matrix):
  for i in range(len(adj_matrix)):
    for j in range(len(adj_matrix[0])):
      if adj_matrix[i][j]:
        return i


def dfs(adj_matrix, vertices):
```

```
n = len(adj_matrix)
current_vertex = 0
tree_matrix = [[0] * n for _ in range(n)]
numeration_matrix = [[0] * n for _ in range(n)]
new = set(range(1, n + 1))
s = Stack()

print("\nСтек:", [x + 1 for x in s.items])
print("Активна вершина: -")
print("Відвідана вершина: -")
print("Номер відвіданої вершини: -")
print("Нові вершини:", new)

start_vertex = getStartVertex(adj_matrix)
visited = [0] * n
s.push(start_vertex)
visited[start_vertex] = 1
found = True
counter = 1
numeration_matrix[start_vertex][counter] = 1

time.sleep(5)

while not s.is_empty():
  vertex = s.items[-1]
  new.discard(vertex + 1)
  print("\nСтек:", [x + 1 for x in s.items])
  print("Активна вершина:", vertex + 1)
  if current_vertex == 0:
    print("Відвідана вершина: - ")
```

```
else:
  print("Відвідана вершина:", vertex + 1)
if found:
  print("Номер відвіданої вершини:", counter)
else:
  print("Номер відвіданої вершини: -")
if not new:
  print("Нові вершини: -")
else:
  print("Нові вершини:", new)
found = False
time.sleep(3)
for i in range(n):
  if not visited[i] and adj_matrix[vertex][i]:
    tree_matrix[vertex][i] = 1
    numeration_matrix[i][counter] = 1
    visited[i] = 1
    s.push(i)
    if i + 1 in new:
      current_vertex = i + 1
    new.discard(i + 1)
    counter += 1
    found = True
    time.sleep(2)
    drawColoredArrows(vertices, vertex, i)
    break
  else:
    current_vertex = 0
if not found:
  s.pop()
```

```
  print("\nСтек:", [x + 1 for x in s.items])
  print("Активна вершина: -")
  print("Відвідана вершина: -")
  print("Номер відвіданої вершини: -")
  print("Нові вершини: -")

  time.sleep(5)

  print("\nМатриця суміжності дерева обходу:")
  for row in tree_matrix:
    print(row)
  print("\nМатриця відповідності номерів вершин:")
  for row in numeration_matrix:
    print(row)
  print("\nОбхід завершено!")


main()
```

Програма для реалізації обходу графа в ширину (BFS)

```
import turtle
import random
import math
import time
from collections import deque


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)
  drawArrows(position, directed_graph_matrix)
  bfs(directed_graph_matrix, position)
  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 * 0.01 - n4 * 0.005 - 0.15
  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):
  n = len(directed_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)
    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 drawArrows(vertices, directed_graph_matrix):
  turtle.speed(0)
  turtle.penup()
  n = len(vertices)
  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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

        turtle.goto(x1, y1)
        turtle.setheading(angle)
        turtle.pendown()
```

```
        turtle.goto(x2, y2)
        turtle.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()

    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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

        turtle.goto(x1, y1)
        turtle.setheading(angle)
        turtle.pendown()
        turtle.goto(x2, y2)
        turtle.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
```

```
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()

    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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

        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.right(150)
        turtle.forward(arrow_size)
        turtle.penup()
```

```
        turtle.goto(x2, y2)
        turtle.left(300)
        turtle.pendown()
        turtle.forward(arrow_size)
        turtle.penup()


  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)) * 20
        y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
        x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
        y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20


        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.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()

    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 * 10, y + math.pi / 1.8 * 10)
        turtle.pendown()
        turtle.circle(10)
        turtle.penup()
        turtle.goto(x + math.pi / 1.6 * 10 - math.pi, y + math.pi / 1.9 * 10 - 1.4 * math.pi)
        turtle.pendown()
        turtle.right(55)
        turtle.backward(10)
        turtle.penup()
        turtle.goto(x + math.pi / 1.6 * 10 - math.pi, y + math.pi / 1.9 * 10 - 1.4 * math.pi)
        turtle.left(90)
        turtle.pendown()
        turtle.backward(10)
        turtle.right(35)
        turtle.penup()


def drawColoredArrows(vertices, i, j):
```

```
turtle.speed(0)
turtle.penup()
turtle.pensize(2)
turtle.color("blue")
arrow_size = 10

if 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)) * 20
  y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
  x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
  y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

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

  turtle.goto(x1, y1)
  turtle.setheading(angle)
  turtle.pendown()
  turtle.goto(x2, y2)
  turtle.right(150)
  turtle.forward(arrow_size)
  turtle.penup()
  turtle.goto(x2, y2)
  turtle.left(300)
  turtle.pendown()
  turtle.forward(arrow_size)
  turtle.penup()
```

```
if 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)) * 20
  y1 += math.sin(math.atan2(y2 - y1, x2 - x1)) * 20
  x2 -= math.cos(math.atan2(y2 - y1, x2 - x1)) * 20
  y2 -= math.sin(math.atan2(y2 - y1, x2 - x1)) * 20

  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.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 getStartVertex(adj_matrix):
  for i in range(len(adj_matrix)):
    for j in range(len(adj_matrix[0])):
      if adj_matrix[i][j]:
        return i


def bfs(adj_matrix, vertices):
  n = len(adj_matrix)
  counter = 0
  num_of_the_visited_vertex = 1
  tree_matrix = [[0] * n for _ in range(n)]
  numeration_matrix = [[0] * n for _ in range(n)]

  start_vertex = getStartVertex(adj_matrix)
  visited = [0] * n
  new = set(range(1, n + 1))
  q = deque()

  print("\nЧерга:", [x + 1 for x in q])
  print("Активна вершина: -")
  print("Відвідана вершина: -")
  print("Номер відвіданої вершини: -")
  print("Нові вершини:", new)

  visited[start_vertex] = 1
  q.append(start_vertex)
```

```
time.sleep(5)

while q:
  vertex = q[0]
  numeration_matrix[vertex][counter] = 1
  new.discard(vertex + 1)
  print("\nЧерга:", [x + 1 for x in q])
  print("Активна вершина:", vertex + 1)
  print("Відвідана вершина: -")
  if vertex == 0:
    print("Номер відвіданої вершини:", num_of_the_visited_vertex)
  else:
    print("Номер відвіданої вершини: -")
  if not new:
    print("Нові вершини: -")
  else:
    print("Нові вершини:", new)
  time.sleep(5)
  for i in range(1, n):
    if not visited[i] and adj_matrix[vertex][i]:
      visited[i] = 1
      num_of_the_visited_vertex += 1
      tree_matrix[vertex][i] = 1
      q.append(i)
      new.discard(i + 1)
      drawColoredArrows(vertices, vertex, i)
      print("\nЧерга:", [x + 1 for x in q])
      print("Активна вершина:", vertex + 1)
      print("Відвідана вершина:", i + 1)
      print("Номер відвіданої вершини:", num_of_the_visited_vertex)
```

```
        if not new:
          print("Нові вершини: -")
        else:
          print("Нові вершини:", new)
        time.sleep(5)
      q.popleft()
      counter += 1

    print("\nЧерга:", [x + 1 for x in q])
    print("Активна вершина: -")
    print("Відвідана вершина: -")
    print("Номер відвіданої вершини: -")
    print("Нові вершини: -")

    time.sleep(5)

    print("\nМатриця суміжності дерева обходу:")
    for row in tree_matrix:
      print(row)
    print("\nМатриця відповідності номерів вершин:")
    for row in numeration_matrix:
      print(row)
    print("\nОбхід завершено!")


main()
```

Матриця суміжності напрямленого графа

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

Матриці суміжності дерев обходу

```
Матриця суміжності дерева обходу:
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

DFS

```
Матриця суміжності дерева обходу:
[0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

BFS

Матриці відповідності нумерації вершин

```
Матриця відповідності номерів вершин:
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 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]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[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, 0, 0, 0, 0, 0, 1]
```

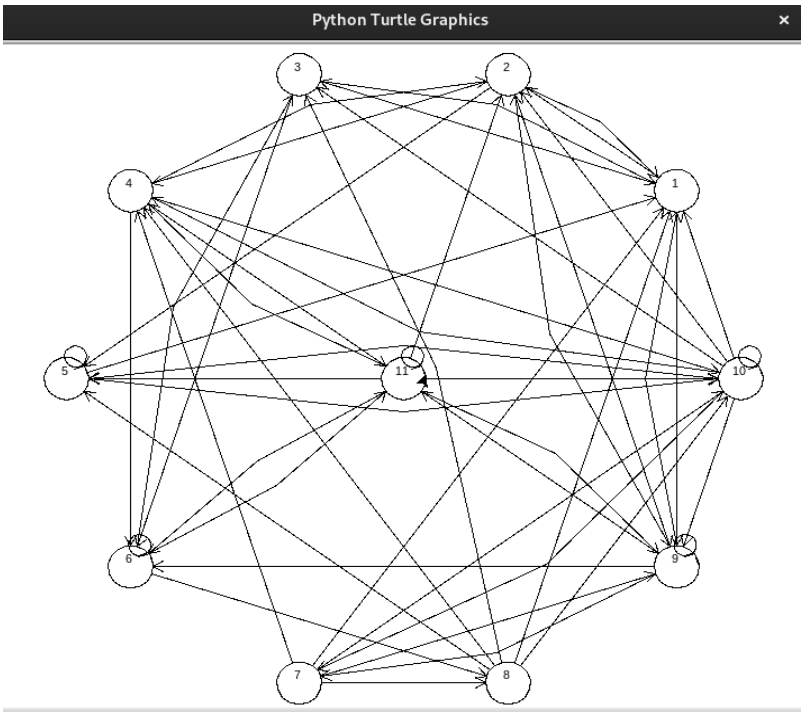
DFS

```
Матриця відповідності номерів вершин:
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 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]
```

BFS

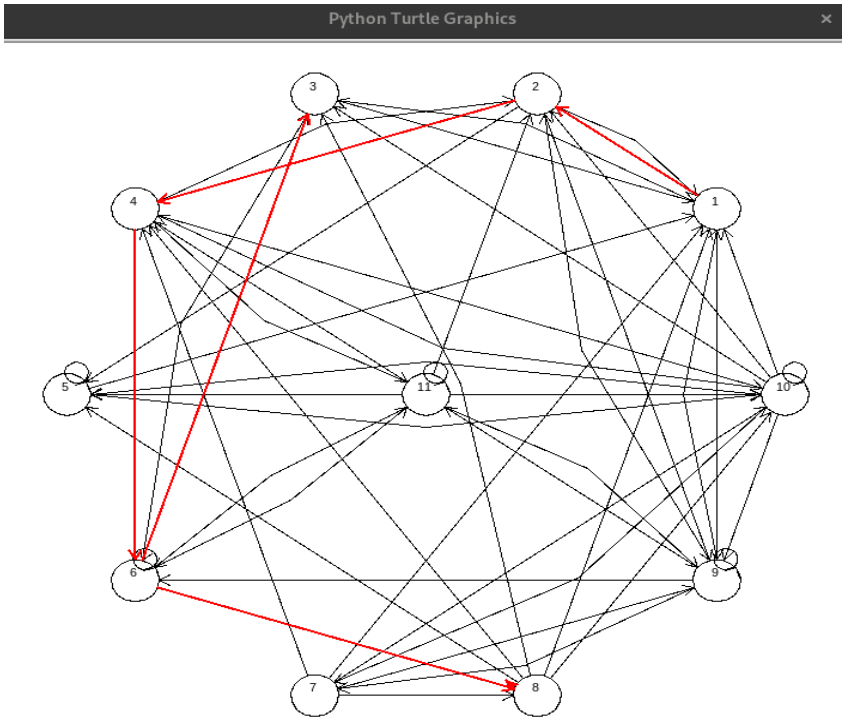
Зображення графа та дерева обходу

DFS, початок обходу



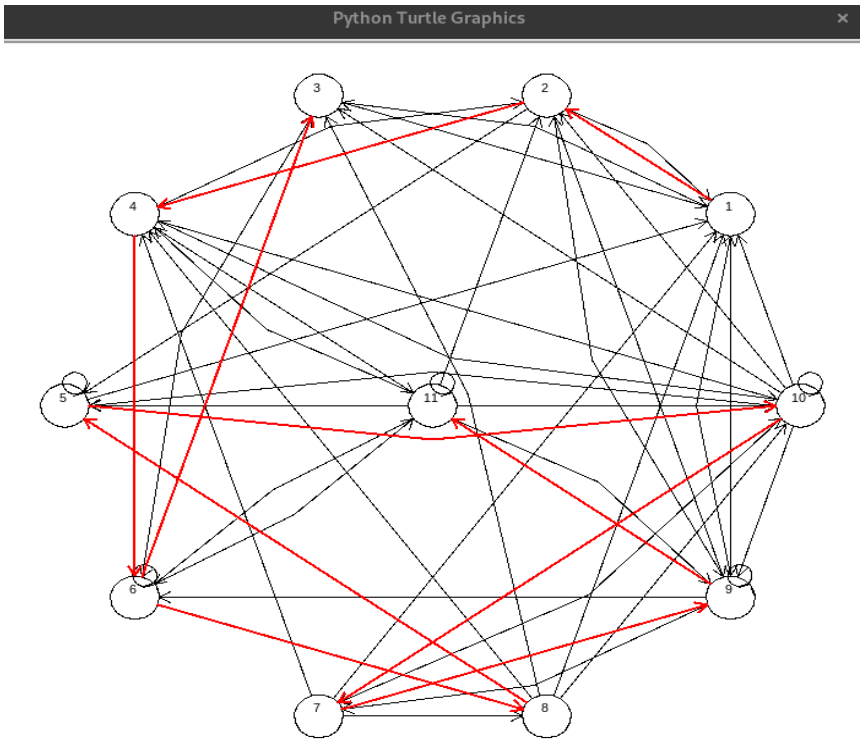

```
Стек: []
Активна вершина: -
Відвідана вершина: -
Номер відвіданої вершини: -
Нові вершини: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

DFS, процес обходу



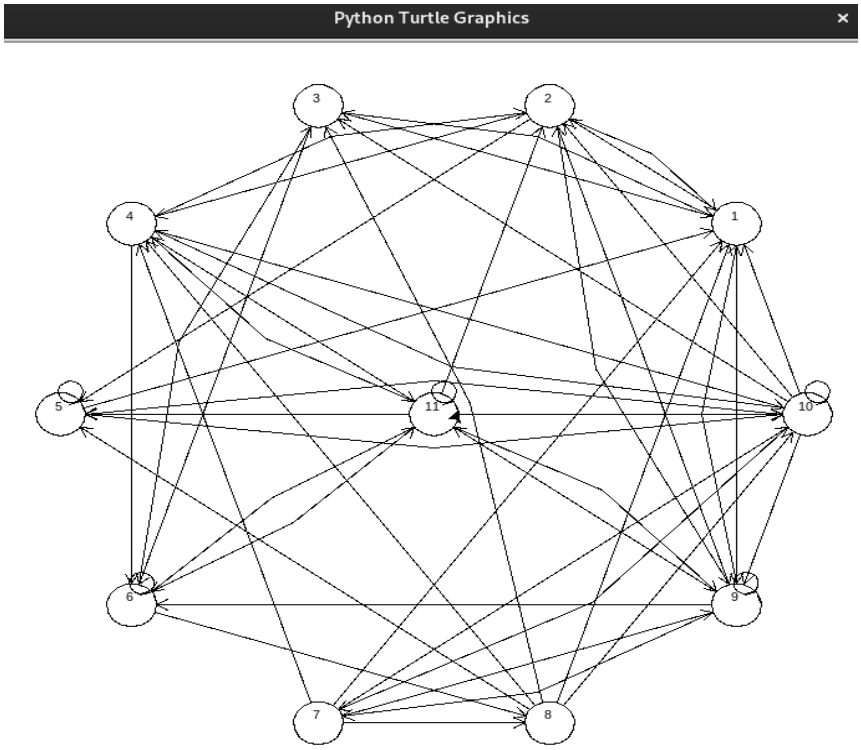

```
Стек: [1, 2, 4, 6, 8]
Активна вершина: 8
Відвідана вершина: 8
Номер відвіданої вершини: 6
Нові вершини: {5, 7, 9, 10, 11}
```

DFS, завершення обходу



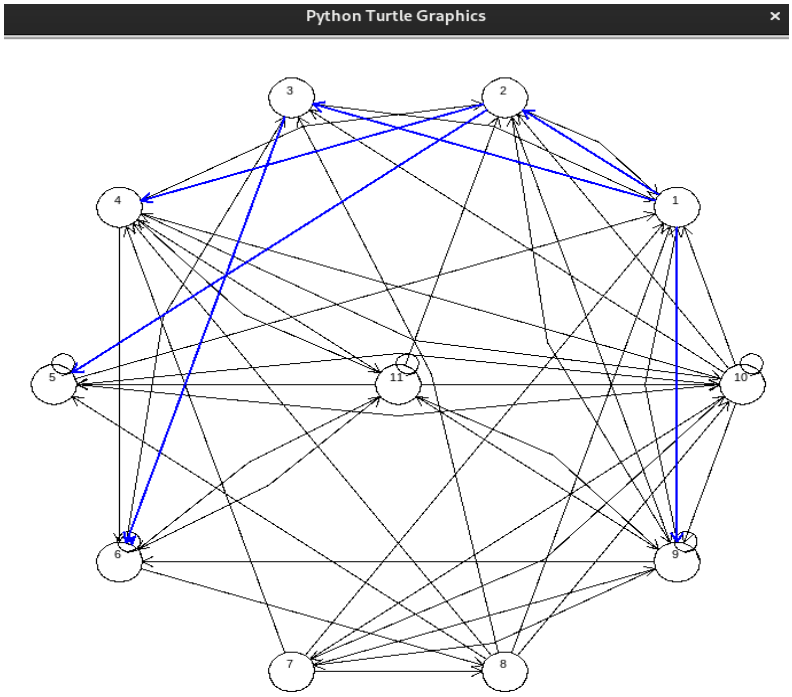

```
Стек: []
Активна вершина: -
Відвідана вершина: -
Номер відвіданої вершини: -
Нові вершини: -
```

BFS, початок обходу



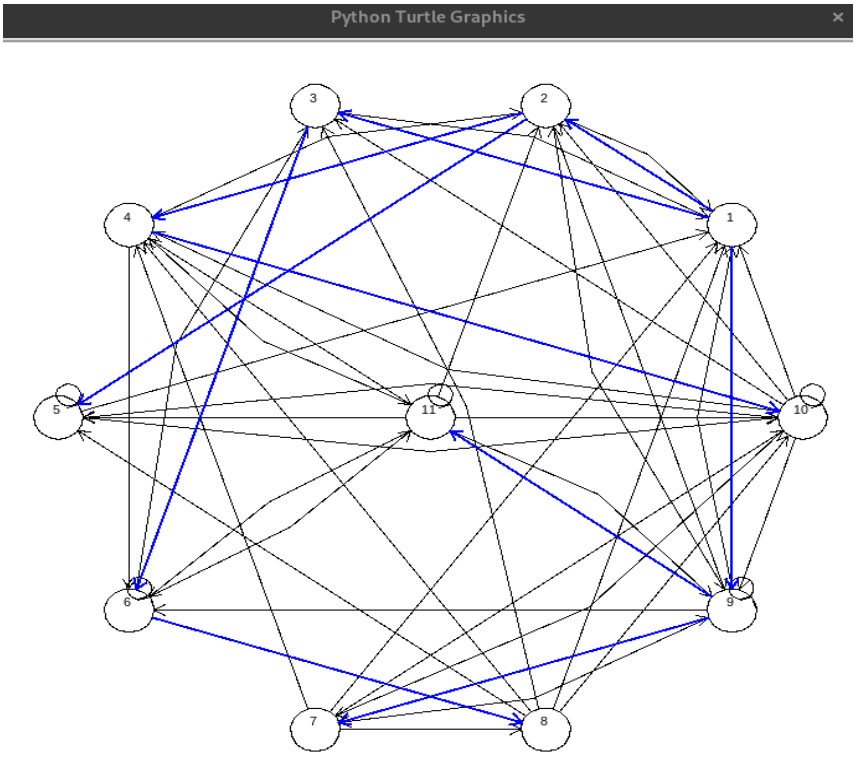

```
Черга: []
Активна вершина: -
Відвідана вершина: -
Номер відвіданої вершини: -
Нові вершини: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

BFS, процес обходу




```
Черга: [3, 9, 4, 5, 6]
Активна вершина: 3
Відвідана вершина: 6
Номер відвіданої вершини: 7
Нові вершини: {7, 8, 10, 11}
```

BFS, завершення обходу




```
Черга: []
Активна вершина: -
Відвідана вершина: -
Номер відвіданої вершини: -
Нові вершини: -
```

Висновок: під час виконання лабораторної роботи я навчився виконувати обхід графа в глибину та ширину, набув навичок використання черги та стеку для зберігання вершин графа. Для реалізації алгоритму BFS було використано чергу, що працює за принципом FIFO («перший прийшов – перший пішов»), а для DFS – стек, що працює за принципом LIFO («останнім прийшов – першим пішов»). Дерево обходу відображено шляхом будування ребер іншого кольору, а протокол обходу виведено в консоль.