**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 IM-43 Сергієнко А. М.

Олексійчук Станіслав Юрійович

номер у списку групи: 22

Київ 2025

***Постановка задачі***

1. Представити у програмi напрямлений i ненапрямлений графи з заданими параметрами:
   * кiлькiсть вершин n;
   * розмiщення вершин;
   * матриця сумiжностi A.
2. Створити програму для формування зображення напрямленого i ненапрямленого графiв у графiчному вiкнi.

***Варіант 22:***

Номер групи: 43

Номер варіанту: 22

n: 4322

Кількість вершин: 12

Формат графа: прямокутник (квадрат)

***Текст програм***

Це завдання було написане на мові програмування Python із використанням графіної бібліотеки tkinter; рішення розділено на певні модулі для логічності та зручності читання:

1. matrix\_print.py – функції створення матриць та їхнього виводу

import math, random

n1 = 4

n2 = 3

n3 = 2

n4 = 2

n = 10 + n3

random.seed(4322)

k = 1.0 - n3 \* 0.02 - n4 \* 0.05 - 0.25

directed\_matrix = [

    [math.floor(random.uniform(0, 2.0) \* k) for \_ in range(n)] for \_ in range(n)

]

undirected\_matrix = [

    [max(directed\_matrix[i][j], directed\_matrix[j][i]) for j in range(n)]

    for i in range(n)

]

labels = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C"]

def matrix\_print(*matrix*, *vertex*, *param*, *title*):

    print(f"\n    {*title*}:\n")

    print("   ", " ".join(labels))

    print("   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    for row in range(*vertex*):

        print(

            f"{*param*[row]} |",

            " ".join(str(*matrix*[row][column]) for column in range(*vertex*)),

        )

1. positions.py – створення масиву позицій вершин (4х4)

def generate\_positions():

    positions = []

    spacing = 120

    offset = 100

    for i in range(3):

        positions.append((offset + i \* spacing, offset))

    for i in range(3):

        positions.append((offset + 3 \* spacing, offset + i \* spacing))

    for i in range(3, 0, -1):

        positions.append((offset + i \* spacing, offset + 3 \* spacing))

    for i in range(3, 0, -1):

        positions.append((offset, offset + i \* spacing))

    return positions

1. draw\_utils.py – функції малювання орієнтованих ребер, ребер, використовуючи різні фігури для належного та зрчуного відтворення в подальшому графа у вікні.

import math

import tkinter as tk

def draw\_arrow(*canvas*, *x1*, *y1*, *x2*, *y2*, *radius*=25):

    dx, dy = *x2* - *x1*, *y2* - *y1*

    dist = math.hypot(dx, dy)

    if dist == 0:

        return

    dx, dy = dx / dist, dy / dist

    start\_x, start\_y = *x1* + dx \* *radius*, *y1* + dy \* *radius*

    end\_x, end\_y = *x2* - dx \* *radius*, *y2* - dy \* *radius*

*canvas*.create\_line(

        start\_x, start\_y, end\_x, end\_y, *arrow*=tk.LAST, *width*=2, *fill*="darkblue"

    )

def draw\_line(*canvas*, *x1*, *y1*, *x2*, *y2*, *radius*=25):

    dx, dy = *x2* - *x1*, *y2* - *y1*

    dist = math.hypot(dx, dy)

    dx, dy = dx / dist, dy / dist

    start\_x, start\_y = *x1* + dx \* *radius*, *y1* + dy \* *radius*

    end\_x, end\_y = *x2* - dx \* *radius*, *y2* - dy \* *radius*

*canvas*.create\_line(start\_x, start\_y, end\_x, end\_y, *width*=2, *fill*="green")

def draw\_arc(*canvas*, *x1*, *y1*, *x2*, *y2*, *radius*=25, *directed*=True):

    dx, dy = *x2* - *x1*, *y2* - *y1*

    dist = math.hypot(dx, dy)

    if dist == 0:

        return

    dx, dy = dx / dist, dy / dist

    start\_x, start\_y = *x1* + dx \* *radius*, *y1* + dy \* *radius*

    end\_x, end\_y = *x2* - dx \* *radius*, *y2* - dy \* *radius*

    mx, my = dy, -dx

    norm = math.hypot(mx, my)

    mx, my = mx / norm, my / norm

    control\_x = (start\_x + end\_x) / 2 + mx \* 60

    control\_y = (start\_y + end\_y) / 2 + my \* 60

*canvas*.create\_line(

        start\_x,

        start\_y,

        control\_x,

        control\_y,

        end\_x,

        end\_y,

*smooth*=True,

*width*=2,

*fill*="darkblue" if *directed* else "green",

*arrow*=tk.LAST if *directed* else None,

    )

def draw\_self\_loop(*canvas*, *x*, *y*, *directed*=True):

    loop\_radius = 20

    canvas\_width, canvas\_height = int(*canvas*["width"]), int(*canvas*["height"])

    center\_x, center\_y = canvas\_width // 2, canvas\_height // 2

    margin = 100

    if abs(*y* - center\_y) < margin:

        if *x* < center\_x:

            bbox, arrow\_start, arrow\_end, angle = (

                (*x* - 55, *y* - loop\_radius, *x* - 15, *y* + loop\_radius),

                (*x* - 23, *y* - loop\_radius + 5),

                (*x* - 20, *y* - loop\_radius + 7),

                45,

            )

        else:

            bbox, arrow\_start, arrow\_end, angle = (

                (*x* + 15, *y* - loop\_radius, *x* + 55, *y* + loop\_radius),

                (*x* + 23, *y* + loop\_radius - 35),

                (*x* + 20, *y* + loop\_radius - 33),

                225,

            )

    elif *y* < center\_y:

        bbox, arrow\_start, arrow\_end, angle = (

            (*x* - loop\_radius, *y* - 55, *x* + loop\_radius, *y* - 15),

            (*x* - 16, *y* - loop\_radius - 5),

            (*x* - 13, *y* - loop\_radius),

            -45,

        )

    else:

        bbox, arrow\_start, arrow\_end, angle = (

            (*x* - loop\_radius, *y* + 15, *x* + loop\_radius, *y* + 55),

            (*x* - loop\_radius + 4, *y* + 25),

            (*x* - loop\_radius + 7, *y* + 20),

            135,

        )

*canvas*.create\_arc(

        bbox,

*start*=angle,

*extent*=270,

*style*=tk.ARC,

*width*=2,

*outline*="darkblue" if *directed* else "green",

    )

    if *directed*:

*canvas*.create\_line(

            \*arrow\_start, \*arrow\_end, *width*=2, *fill*="darkblue", *arrow*=tk.LAST

        )

def is\_crossing\_vertex(*x1*, *y1*, *x2*, *y2*, *positions*, *skip\_indices*, *radius*=25):

    for i, (cx, cy) in enumerate(*positions*):

        if i in *skip\_indices*:

            continue

        num = abs((*y2* - *y1*) \* cx - (*x2* - *x1*) \* cy + *x2* \* *y1* - *y2* \* *x1*)

        den = math.hypot(*y2* - *y1*, *x2* - *x1*)

        if den == 0:

            continue

        dist = num / den

        if dist < *radius*:

            dot1 = (cx - *x1*) \* (*x2* - *x1*) + (cy - *y1*) \* (*y2* - *y1*)

            dot2 = (cx - *x2*) \* (*x1* - *x2*) + (cy - *y2*) \* (*y1* - *y2*)

            if dot1 > 0 and dot2 > 0:

                return True

    return False

def draw\_graph(*canvas*, *positions*):

    radius = 25

    for i, (x, y) in enumerate(*positions*):

*canvas*.create\_oval(

            x - radius, y - radius, x + radius, y + radius, *fill*="lightyellow"

        )

*canvas*.create\_text(x, y, *text*=(i + 1), *font*=("Times New Roman", 12, "bold"))

1. graph\_draw.py – створення графа, використання кнопок для перемикання між напрямленим та ненапрямленим графом

from draw\_utils import (

    draw\_graph,

    draw\_arrow,

    draw\_arc,

    draw\_line,

    is\_crossing\_vertex,

    draw\_self\_loop,

)

from positions import generate\_positions

from matrix\_print import directed\_matrix, n

from tkinter import Button

threshold = 1

positions = generate\_positions()

def create\_graph(*root*, *canvas*):

    def draw\_all(*directed*):

*canvas*.delete("all")

        draw\_graph(*canvas*, positions)

        for i in range(n):

            for j in range(n):

                if directed\_matrix[i][j] == threshold:

                    x1, y1 = positions[i]

                    x2, y2 = positions[j]

                    if i == j:

                        draw\_self\_loop(*canvas*, x1, y1, *directed*=*directed*)

                    elif is\_crossing\_vertex(

                        x1, y1, x2, y2, positions, *skip\_indices*={i, j}

                    ):

                        draw\_arc(*canvas*, x1, y1, x2, y2, *directed*=*directed*)

                    else:

                        if *directed*:

                            draw\_arrow(*canvas*, x1, y1, x2, y2)

                        else:

                            draw\_line(*canvas*, x1, y1, x2, y2)

    draw\_all(*directed*=True)

    def toggle\_graph():

        nonlocal button

        if button.cget("text") == "Switch to Undirected":

            button.config(*text*="Switch to Directed")

            draw\_all(*directed*=False)

        else:

            button.config(*text*="Switch to Undirected")

            draw\_all(*directed*=True)

    button = Button(*root*, *text*="Switch to Undirected", *command*=toggle\_graph)

    button.pack()

1. main.py – вивід матриць суміжності в консоль і графів у вікно

from tkinter import Tk, Canvas

from graph\_draw import create\_graph

from matrix\_print import matrix\_print, directed\_matrix, undirected\_matrix, labels, n

matrix\_print(directed\_matrix, n, labels, "Directed graph")

print("\n    ----------------------\n")

matrix\_print(undirected\_matrix, n, labels, "Undirected graph")

root = Tk()

root.title("Directed/Undirected Graph")

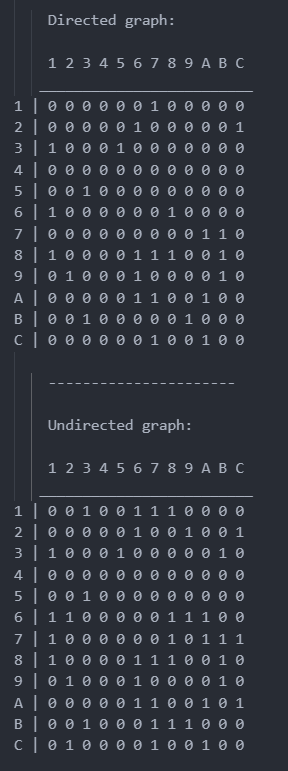
canvas = Canvas(root, *width*=600, *height*=600, *bg*="white")

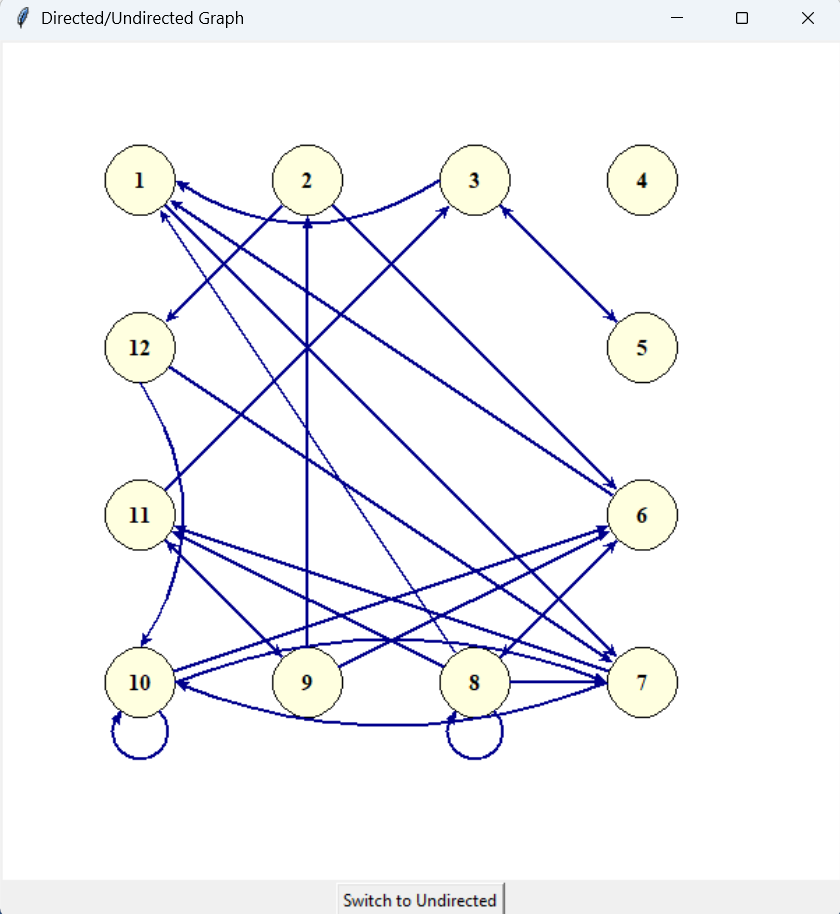
canvas.pack()

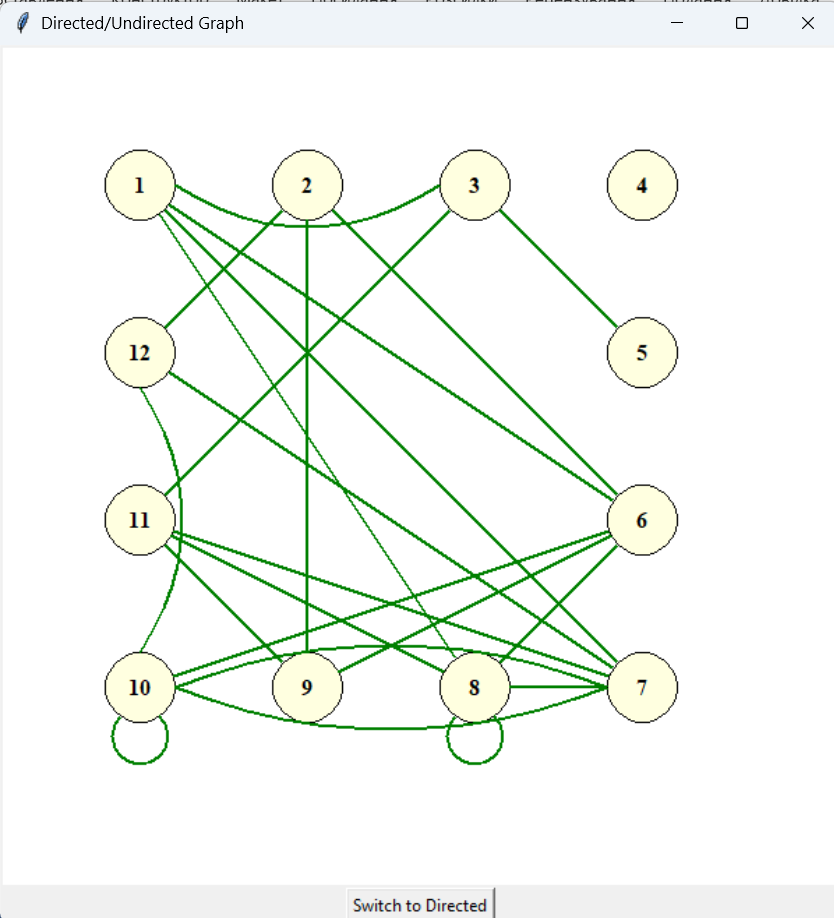
create\_graph(root, canvas)

root.mainloop()

***Результати тестування програми***







***Висновки***

У ході виконання лабораторної роботи було реалізовано генерацію матриці суміжності напрямленого графа та побудовано відповідну матрицю для ненапрямленого графа. Обидві матриці було виведено в консоль.

Також було створено графічне вікно з використанням бібліотеки tkinter, де графи виводяться візуально: вершини розташовано по квадрату, а між ними будуються ребра або напрямлені ребра відповідно до згенерованої матриці суміжності.

У процесі виконання роботи я:

* навчився працювати з матрицями суміжності для напрямлених і ненапрямлених графів;
* закріпив розуміння структури графів;
* покращив навички форматованого виводу інформації у консоль;
* реалізував базову генерацію випадкового графа;
* опанував основи використання бібліотеки tkinter для візуалізації структури графа у графічному інтерфейсі;
* розвинув уважність під час перевірки симетричності та коректності даних.