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-43
Олексійчук Станіслав Юрійович
номер у списку групи: 22

Перевірив:
Сергієнко А. М.

Київ 2025

Постановка задачі

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:
    - кількість вершин n;
    - розміщення вершин;
    - матриця суміжності A.

2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

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)),
        )
```

2) positions.py – створення масиву позицій вершин (4x4)

```python
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
```

3) draw_utils.py – функції малювання орієнтованих ребер, ребер, використовуючи різні фігури для належного та зручного відтворення в подальшому графа у вікні.

```python
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"))
```

4) 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()
```

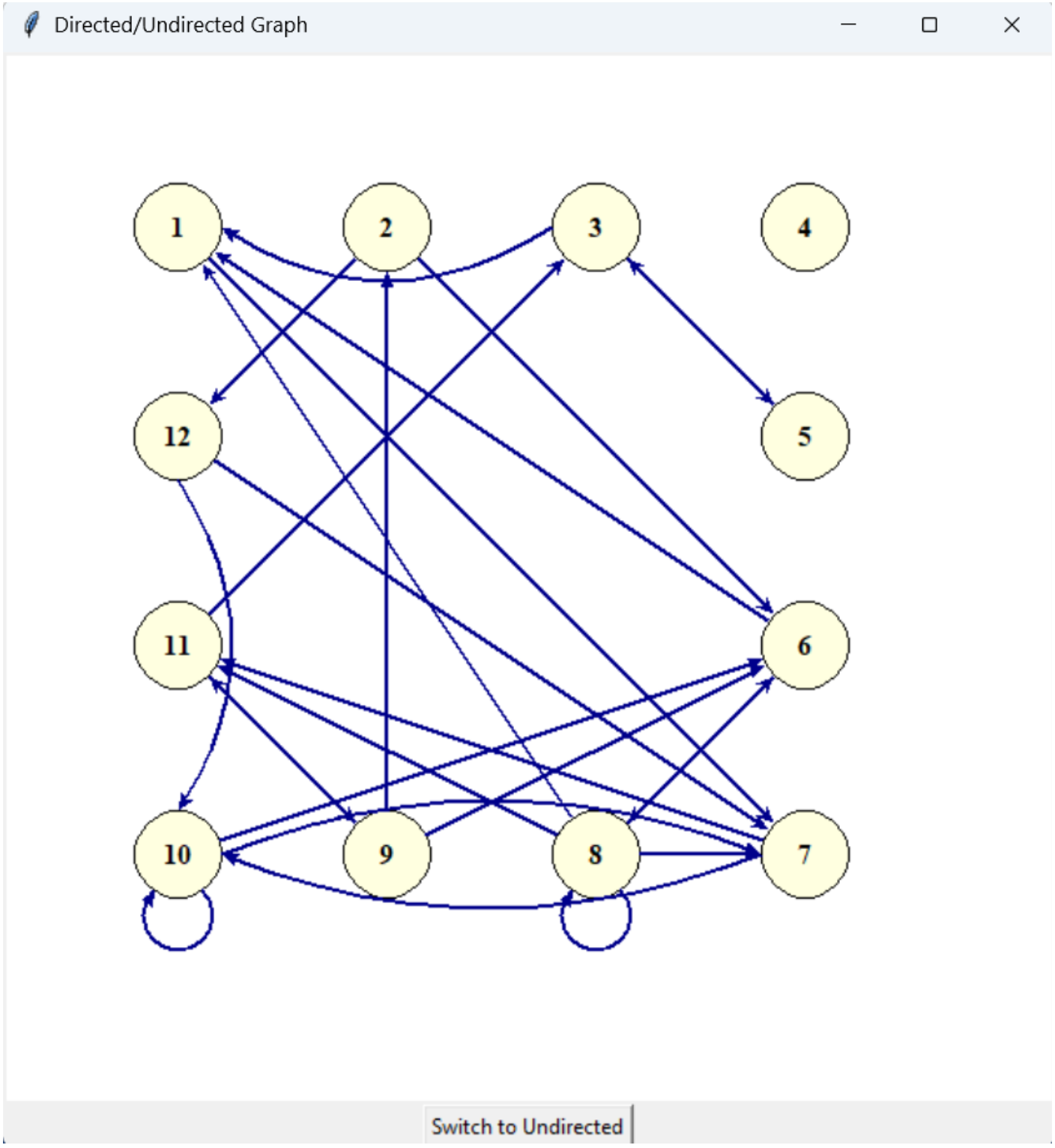
5) main.py – вивід матриць суміжності в консоль і графів у вікно

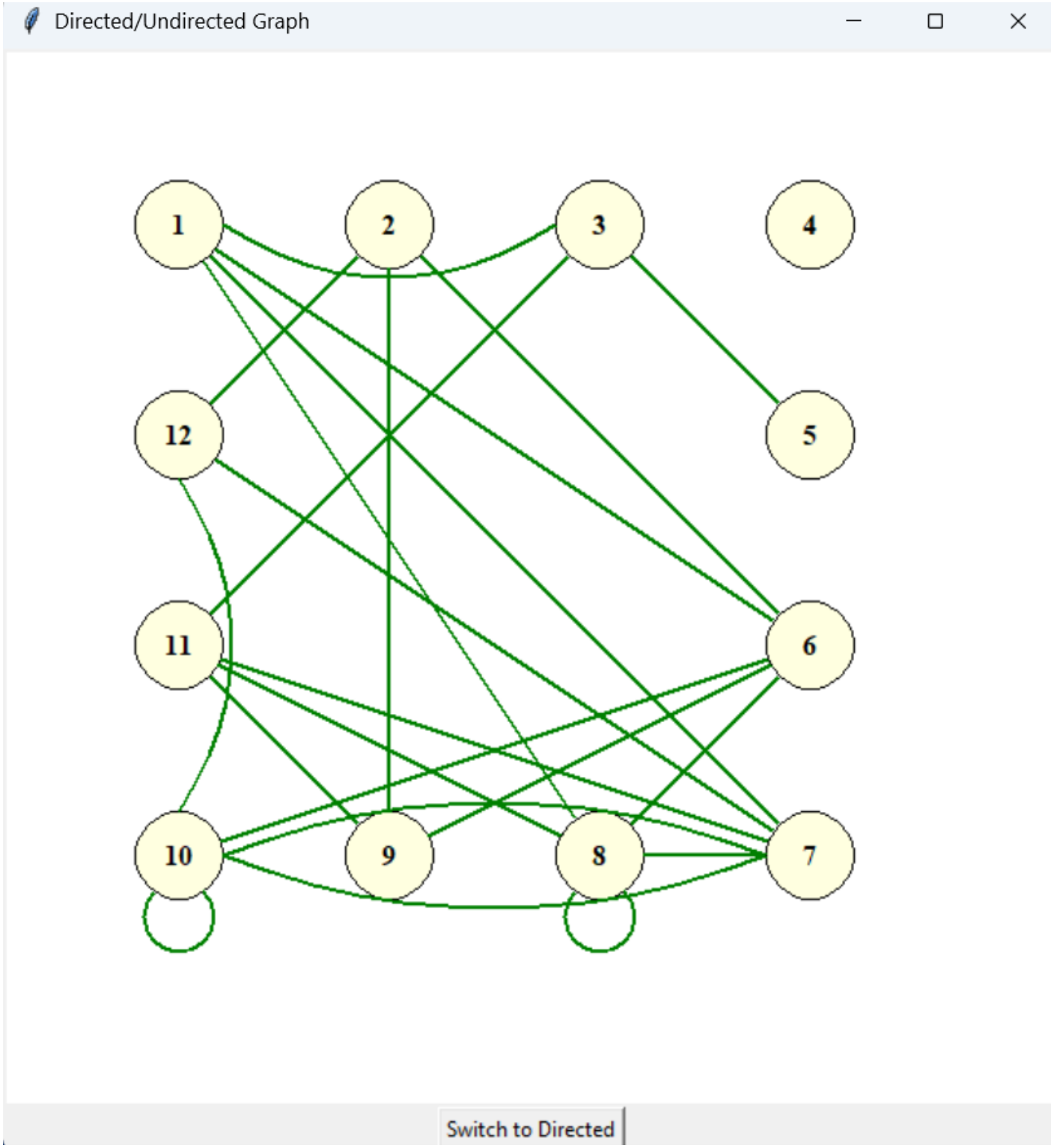
```
from tkinter import Tk, Canvas
from graph_draw import create_graph
from matrix_print import matrix_print, directed_matrix, undirected_matrix,
labels, n

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

```
Directed graph:

  1 2 3 4 5 6 7 8 9 A B C
 ____________________________
1 | 0 0 0 0 0 0 1 0 0 0 0 0
2 | 0 0 0 0 0 1 0 0 0 0 0 1
3 | 1 0 0 0 1 0 0 0 0 0 0 0
4 | 0 0 0 0 0 0 0 0 0 0 0 0
5 | 0 0 1 0 0 0 0 0 0 0 0 0
6 | 1 0 0 0 0 0 0 1 0 0 0 0
7 | 0 0 0 0 0 0 0 0 0 1 1 0
8 | 1 0 0 0 0 1 1 1 0 0 1 0
9 | 0 1 0 0 0 1 0 0 0 0 1 0
A | 0 0 0 0 0 1 1 0 0 1 0 0
B | 0 0 1 0 0 0 0 0 1 0 0 0
C | 0 0 0 0 0 0 1 0 0 1 0 0

  ----------------------

  Undirected graph:

  1 2 3 4 5 6 7 8 9 A B C
 ____________________________
1 | 0 0 1 0 0 1 1 1 0 0 0 0
2 | 0 0 0 0 0 1 0 0 1 0 0 1
3 | 1 0 0 0 1 0 0 0 0 0 1 0
4 | 0 0 0 0 0 0 0 0 0 0 0 0
5 | 0 0 1 0 0 0 0 0 0 0 0 0
6 | 1 1 0 0 0 0 0 1 1 1 0 0
7 | 1 0 0 0 0 0 0 1 0 1 1 1
8 | 1 0 0 0 0 1 1 1 0 0 1 0
9 | 0 1 0 0 0 1 0 0 0 0 1 0
A | 0 0 0 0 0 1 1 0 0 1 0 1
B | 0 0 1 0 0 0 1 1 1 0 0 0
C | 0 1 0 0 0 0 1 0 0 1 0 0
```


Directed/Undirected Graph
1
2
3
4
12
5
11
6
10
9
8
7
Switch to Undirected


Directed/Undirected Graph
1
2
3
4
12
5
11
6
10
9
8
7
Switch to Directed

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.