

**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-42
Федоренко Іван Русланович
номер у списку групи: 29

Перевірив:

Сергієнко А. М.

Київ 2025

Завдання

Лабораторна робота 3.

Графічне представлення графів

Мета лабораторної роботи

Метою лабораторної роботи №3 «Графічне представлення графів» є набуття практичних навичок представлення графів у комп'ютері та ознайомлення з принципами роботи ОС.

Постановка задачі

1. Представити у програмі напрямлений і ненаправлений графи з заданими параметрами:

- кількість вершин n ;
- розміщення вершин;
- матриця суміжності A .

2. Створити програму для формування зображення напрямленого і ненаправленого 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 (у мене група 42), а n_3n_4 — десяткові цифри номера варіанту (мій варіант 29).

Кількість вершин n дорівнює $10 + n_3$.

Розміщення вершин:

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

Наприклад, при $n_4 = 9$ розміщення вершин прямокутником з вершиною в центрі

Матриця суміжності A для напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$ — детальніше див. с. 12;

- 2) матриця розміром $n \times n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$;
- 3) обчислюється коефіцієнт $n3 * 0.02 - n4 * 0.005 - 0.25$
- 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.

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

$$a(dir\ i,j) = 1 \Rightarrow (a_{undir\ i,j}) = 1, a_{undir\ j,i} = 1.$$

Наприклад, якщо запрограмувати додаткові функції `randm` та `mulmr`, у програмі мовою C генерація матриці A_{dir} напрямленого графа може виглядати так:

```
srand(n1n2n3n4);
```

```
T = randm(n);
```

```
k = 1.0 - n3*0.02 - n4*0.005 - 0.25;
```

```
A = mulmr(T, k);
```

Тут `randm(n)` — розроблена функція, яка формує матрицю розміром nn , що складається з випадкових чисел у діапазоні $(0, 2.0)$; `mulmr(T, k)` — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1.

Код програми:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def generate_adjacency_matrix(n, variant_number):
    """Generate directed adjacency matrix based on variant number"""
    np.random.seed(variant_number)
    T = np.random.random((n, n)) * 2.0

    n3 = 2
    n4 = 9
    k = 1.0 - n3 * 0.02 - n4 * 0.005 - 0.25

    A = np.zeros((n, n), dtype=int)
    for i in range(n):
        for j in range(n):
            A[i, j] = 1 if T[i, j] * k >= 1.0 else 0

    return A
```

```

def get_undirected_matrix(directed_matrix):
    """Convert directed adjacency matrix to undirected"""
    n = directed_matrix.shape[0]
    undirected_matrix = np.zeros((n, n), dtype=int)

    for i in range(n):
        for j in range(n):
            if directed_matrix[i, j] == 1:
                undirected_matrix[i, j] = 1
                undirected_matrix[j, i] = 1

    return undirected_matrix

def get_vertex_positions(n, n4):
    """Get vertex positions based on n4 value"""
    positions = np.zeros((n, 2))

    if n4 in [8, 9]:
        width, height = 12, 8
        positions[n-1] = [0, 0]

        perimeter_vertices = n - 1
        sides = [0, 0, 0, 0]
        remaining = perimeter_vertices - 4

        for i in range(remaining):
            sides[i % 4] += 1

        vertex_index = 0

        positions[vertex_index] = [-width/2, height/2]
        vertex_index += 1

        for i in range(sides[0]):
            x = -width/2 + (i+1) * width / (sides[0]+1)
            positions[vertex_index] = [x, height/2]
            vertex_index += 1

        positions[vertex_index] = [width/2, height/2]
        vertex_index += 1

        for i in range(sides[1]):
            y = height/2 - (i+1) * height / (sides[1]+1)
            positions[vertex_index] = [width/2, y]
            vertex_index += 1

        positions[vertex_index] = [width/2, -height/2]
        vertex_index += 1

```

```

    for i in range(sides[2]):
        x = width/2 - (i+1) * width / (sides[2]+1)
        positions[vertex_index] = [x, -height/2]
        vertex_index += 1

    positions[vertex_index] = [-width/2, -height/2]
    vertex_index += 1

    for i in range(sides[3]):
        y = -height/2 + (i+1) * height / (sides[3]+1)
        positions[vertex_index] = [-width/2, y]
        vertex_index += 1

    return positions

def draw_edge(ax, start, end, is_directed=True, color='blue', linewidth=1.5):
    """Draw an edge between two vertices"""
    dx = end[0] - start[0]
    dy = end[1] - start[1]

    dist = np.sqrt(dx**2 + dy**2)
    if dist < 0.001:
        return

    vertex_radius = 0.5
    ratio = vertex_radius / dist
    start_x = start[0] + dx * ratio
    start_y = start[1] + dy * ratio
    end_x = end[0] - dx * ratio
    end_y = end[1] - dy * ratio

    rad = 0.2

    if is_directed:
        arrow = patches.FancyArrowPatch(
            (start_x, start_y), (end_x, end_y),
            arrowstyle='->',
            color=color,
            linewidth=linewidth,
            connectionstyle=f'arc3,rad={rad}',
            mutation_scale=15
        )
    else:
        arrow = patches.FancyArrowPatch(
            (start_x, start_y), (end_x, end_y),
            arrowstyle='-',
            color=color,
            linewidth=linewidth,
            connectionstyle=f'arc3,rad={rad}'

```

```

    )

    ax.add_patch(arrow)

def draw_graph(directed_matrix, positions, is_directed=True):
    """Draw a graph based on adjacency matrix and vertex positions"""
    n = directed_matrix.shape[0]

    fig, ax = plt.subplots(figsize=(12, 10))

    rect = patches.Rectangle((-6, -4), 12, 8, linewidth=1, edgecolor='gray',
                             facecolor='none', linestyle='--')
    ax.add_patch(rect)

    for i in range(n):
        for j in range(n):
            if directed_matrix[i, j] == 1:
                draw_edge(ax, positions[i], positions[j], is_directed, color='blue')

    for i, pos in enumerate(positions):
        circle = plt.Circle(pos, 0.5, fill=True, color='lightblue', edgecolor='blue')
        ax.add_patch(circle)

        ax.text(pos[0], pos[1], str(i+1), horizontalalignment='center',
                verticalalignment='center', fontsize=10, color='black',
                fontweight='bold')

    ax.set_aspect('equal')
    margin = 2
    ax.set_xlim(min(positions[:, 0])-margin, max(positions[:, 0])+margin)
    ax.set_ylim(min(positions[:, 1])-margin, max(positions[:, 1])+margin)

    graph_type = "Directed" if is_directed else "Undirected"
    plt.title(f"{graph_type} Graph - {n} vertices")

    plt.axis('off')

def print_matrix(matrix, title):
    """Print the adjacency matrix in a readable format"""
    print(f"\n{title}:")
    for row in matrix:
        print(" ".join(map(str, row)))

def main():
    variant_number = 4229
    n3 = 2
    n4 = 9
    n = 10 + n3

```

```

print(f"Variant number: {variant_number}")
print(f"Number of vertices n = 10 + {n3} = {n}")
print(f"Vertex placement: Rectangular with vertex in center (n4 = {n4})")

directed_matrix = generate_adjacency_matrix(n, variant_number)
undirected_matrix = get_undirected_matrix(directed_matrix)

print_matrix(directed_matrix, f"Directed Graph Adjacency Matrix ({n}x{n})")
print_matrix(undirected_matrix, f"Undirected Graph Adjacency Matrix ({n}x{n})")

positions = get_vertex_positions(n, n4)

draw_graph(directed_matrix, positions, is_directed=True)
plt.savefig('directed_graph.png')

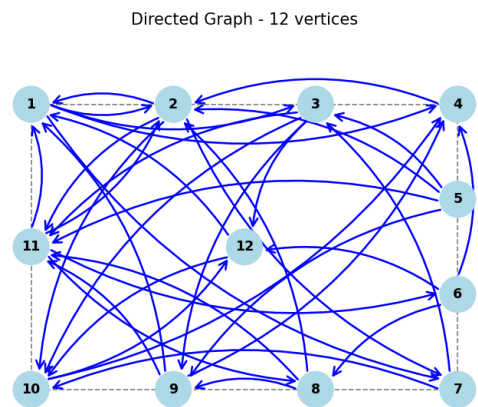
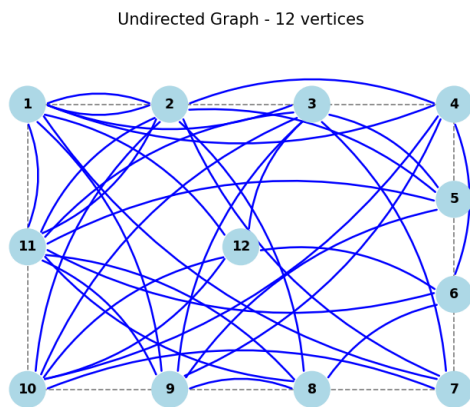
draw_graph(directed_matrix, positions, is_directed=False)
plt.savefig('undirected_graph.png')

plt.show()

if __name__ == "__main__":
    main()

```

Вивід:



Результати тестування:

```
Variant number: 4229
Number of vertices n = 10 + 2 = 12
Vertex placement: Rectangular with vertex in center (n4 = 9)

Directed Graph Adjacency Matrix (12x12):
0 1 1 1 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 0 0 1 1 1 1
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 1 0 0 0 0 0 1 0 1 0
0 0 0 1 0 0 0 1 0 0 0 1
0 0 1 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0 1 0
1 0 0 1 0 0 0 0 1 0 1 0
0 0 0 1 0 0 0 0 0 1 0 1
1 1 0 0 0 1 0 1 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0

Undirected Graph Adjacency Matrix (12x12):
0 1 1 1 0 0 1 0 1 0 1 1
1 0 0 1 1 0 1 1 0 1 1 0
1 0 0 0 1 0 1 0 1 1 1 1
1 1 0 0 0 1 0 0 1 1 0 0
0 1 1 0 0 0 0 0 1 0 1 0
0 0 0 1 0 0 0 1 0 0 1 1
1 1 1 0 0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0 1 0 1 0
1 0 1 1 1 0 0 1 1 0 1 0
0 1 1 1 0 0 1 0 0 1 0 1
1 1 1 0 1 1 0 1 1 0 0 0
1 0 1 0 0 1 0 0 0 1 0 0
```

Висновки:

У ході виконання лабораторної роботи було реалізовано програму для створення та графічного зображення напрямленого й ненаправленого графів із заданими параметрами. Було сформовано матриці суміжності на основі генерації випадкових чисел з урахуванням номера варіанту, а також реалізовано алгоритм візуалізації графів у графічному вікні з використанням мови програмування Python та бібліотеки matplotlib.

Виконана робота сприяла закріпленню практичних навичок у представленні графів у комп'ютерних програмах, формуванні матриць суміжності та застосуванні базових графічних примітивів.