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-43
Олексійчук Станіслав Юрійович
номер у списку групи: 22

Перевірив:
Сергієнко А. М.

Київ 2025

Постановка задачі

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

   Відмінність: коефіцієнт $k = 1.0 - n_3 \cdot 0.01 - n_4 \cdot 0.005 - 0.15$;

   Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

   1) встановлюється параметр (seed) генератора випадкових чисел, рів- не номеру варіанту n1n2n3n4;
   2) матриця розміром n·n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
   3) обчислюється коефіцієнт k= $1.0 - n_3 \cdot 0.01 - n_4 \cdot 0.005 - 0.15$, кожен елемент матриці множиться на коефіцієнт k;

   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).
   - обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;
   - при обході враховувати порядок нумерації;
   - 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 об- ходу виводити покроково у процесі виконання обходу графа. Це можна виконати одним із двох способів:
   - або виділяти іншим кольором ребра графа;
   - або будувати дерево обходу поряд із графом.
4. Зміну статусів вершин у процесі обходу продемонструвати зміною ко- льорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу у графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжува- ти обхід з невідвіданої вершини з найменшим номером, яка має щонай- менше одну вихідну дугу.

Варіант 22:

Номер групи: 43

Номер варіанту: 22

Seed: 4322

Кількість вершин: 12

Формат графа: прямокутник (квадрат)

Текст програм

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

1) matrix_print.py – функції створення матриць та їхнього виводу

```
import random

n1 = 4
n2 = 3
n3 = 2
n4 = 2

n = 10 + n3

random.seed(4322)

k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.15

directed_matrix = [
    [1 if random.uniform(0, 2.0) * k >= 1.0 else 0 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(param[:vertex]))
    print("   " + "_" * (vertex * 2 + 1))
    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)

```
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) graph_traversal.py – клас для створення функцій обходу графів методами BFS/DFS, скасування (reset), виведення матриці суміжності дерева обходу та списку (вектора) відповідності номерів вершин і їх нової нумерації.

```
from collections import deque


class GraphTraversal:
    def __init__(self, matrix):
        self.matrix = matrix
        self.n = len(matrix)
        self.reset()

    def reset(self):
        self.visited = [False] * self.n
        self.parent = [-1] * self.n
        self.queue = deque()
        self.stack = []
        self.current_vertex = -1
        self.traversal_type = None
        self.traversal_order = []
        self.traversal_tree = []
        self.started = False
        self.completed = False

    def find_start_vertex(self):
        for i in range(self.n):
            if any(self.matrix[i]) and not self.visited[i]:
                return i
```

```
        return -1

    def find_unvisited_vertex(self):
        for i in range(self.n):
            if not self.visited[i] and any(self.matrix[i]):
                return i
        return -1

    def init_bfs(self):
        self.reset()
        self.traversal_type = "BFS"
        start_vertex = self.find_start_vertex()
        if start_vertex != -1:
            self.queue.append(start_vertex)
            self.visited[start_vertex] = True
            self.current_vertex = start_vertex
            self.traversal_order.append(start_vertex)
            self.started = True
            self.completed = False
            return start_vertex
        return -1

    def init_dfs(self):
        self.reset()
        self.traversal_type = "DFS"
        start_vertex = self.find_start_vertex()
        if start_vertex != -1:
            self.stack.append(start_vertex)
            self.visited[start_vertex] = True
            self.current_vertex = start_vertex
            self.traversal_order.append(start_vertex)
            self.started = True
            self.completed = False
            return start_vertex
        return -1

    def get_unvisited_neighbors(self, vertex):
        return [
            i for i in range(self.n) if self.matrix[vertex][i] and not
self.visited[i]
        ]

    def next_step(self):
        if not self.started or self.completed:
            return None
```

```
    if self.traversal_type == "BFS":
        return self.bfs_step()
    else:
        return self.dfs_step()

def bfs_step(self):
    if not self.queue:
        start_vertex = self.find_unvisited_vertex()
        if start_vertex == -1:
            self.completed = True
            if self.completed:
                self.print_traversal_tree_matrix()
                self.print_renumbering()
            return None
        self.queue.append(start_vertex)
        self.visited[start_vertex] = True
        self.traversal_order.append(start_vertex)
        self.current_vertex = start_vertex
        return start_vertex

    current = self.queue[0]
    self.current_vertex = current

    neighbors = self.get_unvisited_neighbors(current)

    if neighbors:
        neighbor = neighbors[0]
        self.visited[neighbor] = True
        self.parent[neighbor] = current
        self.traversal_tree.append((current, neighbor))
        self.queue.append(neighbor)
        self.traversal_order.append(neighbor)
        self.current_vertex = neighbor
        return neighbor
    else:
        self.queue.popleft()
        return current

def dfs_step(self):
    if not self.stack:
        start_vertex = self.find_unvisited_vertex()
        if start_vertex == -1:
            self.completed = True
            if self.completed:
                self.print_traversal_tree_matrix()
                self.print_renumbering()
```

```
                return None
            self.stack.append(start_vertex)
            self.visited[start_vertex] = True
            self.traversal_order.append(start_vertex)
            self.current_vertex = start_vertex
            return start_vertex

        current = self.stack[-1]
        self.current_vertex = current

        neighbors = self.get_unvisited_neighbors(current)

        if neighbors:
            neighbor = neighbors[0]
            self.visited[neighbor] = True
            self.parent[neighbor] = current
            self.traversal_tree.append((current, neighbor))
            self.stack.append(neighbor)
            self.traversal_order.append(neighbor)
            self.current_vertex = neighbor
            return neighbor
        else:
            self.stack.pop()
            return current

    def get_vertex_color(self, vertex):
        if vertex == self.current_vertex:
            return "red"
        if self.visited[vertex]:
            return "lightgreen"
        return "lightyellow"

    def print_traversal_tree_matrix(self):
        tree_matrix = [[0] * self.n for _ in range(self.n)]

        for child in range(self.n):
            parent = self.parent[child]
            if parent != -1:
                tree_matrix[parent][child] = 1

        print("\n=== Traversal Tree Adjacency Matrix ===")
        print("   " + " ".join(f"{i+1:2d}" for i in range(self.n)))
        print("  " + "-" * (self.n * 3 + 1))

        for i in range(self.n):
            if self.visited[i] or any(tree_matrix[i]):
```

```
                row_str = "  ".join(f"{val:2d}" for val in
tree_matrix[i])
                print(f"{i+1:2d}|{row_str}")

    def print_renumbering(self):
        print("\n=== Vertex Renumbering ===")
        print("Format: [original number] -> [new number in traversal
order]")

        renumbering = {}
        for new_num, original_num in enumerate(self.traversal_order, 1):
            renumbering[original_num] = new_num

        for i in range(self.n):
            if i in renumbering:
                print(f"{i+1:2d} -> {renumbering[i]:2d}")
            else:
                print(f"{i+1:2d} -> Not reached in traversal")
```

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

```
import math
import tkinter as tk


def draw_arrow(canvas, x1, y1, x2, y2, radius=25, color="darkblue",
width=2):
    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=width,
fill=color
    )


def draw_line(canvas, x1, y1, x2, y2, radius=25, color="green", width=2):
    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=width,
fill=color)


def draw_arc(
    canvas, x1, y1, x2, y2, radius=25, directed=True, color="darkblue",
width=2
):
    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=width,
        fill=color,
        arrow=tk.LAST if directed else None,
    )


def draw_self_loop(canvas, x, y, directed=True, color="darkblue", width=2):
    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=width,
        outline=color,
    )

    if directed:
        canvas.create_line(
            *arrow_start,     *arrow_end,     width=width,     fill=color,
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
```

5) graph_draw.py – створення графа, використання кнопок для перемикання між напрямленим, ненапрямленим графом, методами обходу BFS, DFS, перемикання між кроками, скасування (reset) та показу матриці суміжності дерева обходу й списку (вектора) відповідності номерів вершин і їх нової нумерації, набутої в процесі обходу на той момент.

```
from draw_utils import *
from positions import generate_positions
from matrix_print import directed_matrix, n
from tkinter import Button, Tk, Canvas, Frame, messagebox, Label
from graph_traversal import GraphTraversal

threshold = 1
positions = generate_positions()


def create_graph_window():
    graph_window = Tk()
    graph_window.title("Graph Visualization")

    canvas = Canvas(graph_window, width=600, height=600)
    canvas.pack()

    info_frame = Frame(graph_window)
    info_frame.pack(pady=5)

    status_label = Label(
        info_frame, text="Status: Ready for traversal", font=("Arial", 12)
    )
    status_label.pack()
```

```
    traversal_info = Label(info_frame, text="", font=("Arial", 10))
    traversal_info.pack()

    button_frame = Frame(graph_window)
    button_frame.pack()

    traversal = GraphTraversal(directed_matrix)

    def draw_all(directed=True):
        canvas.delete("all")

        for i in range(n):
            for j in range(n):
                if directed_matrix[i][j] == threshold:
                    if not directed and directed_matrix[j][i] == threshold
and j < i:
                        continue

                    x1, y1 = positions[i]
                    x2, y2 = positions[j]

                    edge_color = "gray"
                    width = 1
                    if (i, j) in traversal.traversal_tree:
                        edge_color = "red"
                        width = 3

                    if i == j:
                        draw_self_loop(
                            canvas,
                            x1,
                            y1,
                            directed=directed,
                            color=edge_color,
                            width=width,
                        )
                    elif is_crossing_vertex(
                        x1, y1, x2, y2, positions, skip_indices={i, j}
                    ):
                        draw_arc(
                            canvas,
                            x1,
                            y1,
                            x2,
                            y2,
```

```
                            directed=directed,
                            color=edge_color,
                            width=width,
                        )
                    elif directed:
                        if directed_matrix[j][i] == threshold:
                            draw_arc(
                                canvas,
                                x1,
                                y1,
                                x2,
                                y2,
                                directed=directed,
                                color=edge_color,
                                width=width,
                            )
                        else:
                            draw_arrow(
                                canvas, x1, y1, x2, y2, color=edge_color,
width=width
                            )
                    else:
                        draw_line(canvas, x1, y1, x2, y2, color=edge_color,
width=width)

        radius = 25
        for i, (x, y) in enumerate(positions):
            color = traversal.get_vertex_color(i)
            canvas.create_oval(
                x - radius,
                y - radius,
                x + radius,
                y + radius,
                fill=color,
                outline="black",
                width=2,
            )
            canvas.create_text(
                x,
                y,
                text=str(i + 1),
                font=("Times New Roman", 12, "bold"),
                fill="black",
            )

        status_text = "Status: "
```

```
        if not traversal.started:
            status_text += "Ready for traversal"
            traversal_info.config(text="")
        elif traversal.completed:
            status_text += "Traversal completed"
            order_text = "Traversal order: " + " → ".join(
                [str(v + 1) for v in traversal.traversal_order]
            )
            traversal_info.config(text=order_text)
            status_label.config(text=status_text)

            messagebox.showinfo(
                "Traversal Complete",
                "Graph traversal completed!\nTraversal tree adjacency
matrix and vertex renumbering are displayed in the console.",
            )
        else:
            if traversal.traversal_type == "BFS":
                status_text += (
                    f"BFS traversal - current vertex:
{traversal.current_vertex + 1}"
                )
            else:
                status_text += (
                    f"DFS traversal - current vertex:
{traversal.current_vertex + 1}"
                )

            if traversal.traversal_order:
                order_text = "Traversal order: " + " → ".join(
                    [str(v + 1) for v in traversal.traversal_order]
                )
                traversal_info.config(text=order_text)

        status_label.config(text=status_text)

    def toggle_graph():
        if button_mode.cget("text") == "Switch to Undirected":
            button_mode.config(text="Switch to Directed")
            draw_all(directed=False)
        else:
            button_mode.config(text="Switch to Undirected")
            draw_all(directed=True)

    def start_bfs():
        if traversal.init_bfs() == -1:
```

```
            messagebox.showwarning(
                "No Start Vertex", "No vertex with outgoing edges found!"
            )
        draw_all()

    def start_dfs():
        if traversal.init_dfs() == -1:
            messagebox.showwarning(
                "No Start Vertex", "No vertex with outgoing edges found!"
            )
        draw_all()

    def next_step():
        if not traversal.started:
            messagebox.showwarning("Not Started", "Please start BFS or DFS
first!")
            return
        traversal.next_step()
        draw_all()

    def reset_traversal():
        traversal.reset()
        traversal_info.config(text="")
        draw_all()

    button_mode = Button(
        button_frame, text="Switch to Undirected", command=toggle_graph
    )
    button_mode.pack(side="left", padx=5)

    Button(button_frame, text="BFS", command=start_bfs).pack(side="left",
padx=5)
    Button(button_frame, text="DFS", command=start_dfs).pack(side="left",
padx=5)
    Button(button_frame,                text="Next                Step",
command=next_step).pack(side="left", padx=5)
    Button(button_frame, text="Reset", command=reset_traversal).pack(
        side="left", padx=5
    )

    def show_matrix_and_renumbering():
        if not traversal.completed and traversal.started:
            traversal.print_traversal_tree_matrix()
            traversal.print_renumbering()
        elif not traversal.started:
```

```
            messagebox.showinfo("Info", "First start traversal (BFS or
DFS).")
        else:
            traversal.print_traversal_tree_matrix()
            traversal.print_renumbering()

    Button(
        button_frame,
        text="Show Matrix & Renumbering",
        command=show_matrix_and_renumbering,
    ).pack(side="left", padx=5)

    draw_all(directed=True)
    graph_window.mainloop()
```

6) main.py – вивід матриць суміжності та виконання головної графічної функції

```
from graph_draw import create_graph_window
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")

create_graph_window()
```

Результати тестування програми

1. Матриця суміжності напрямленого графа

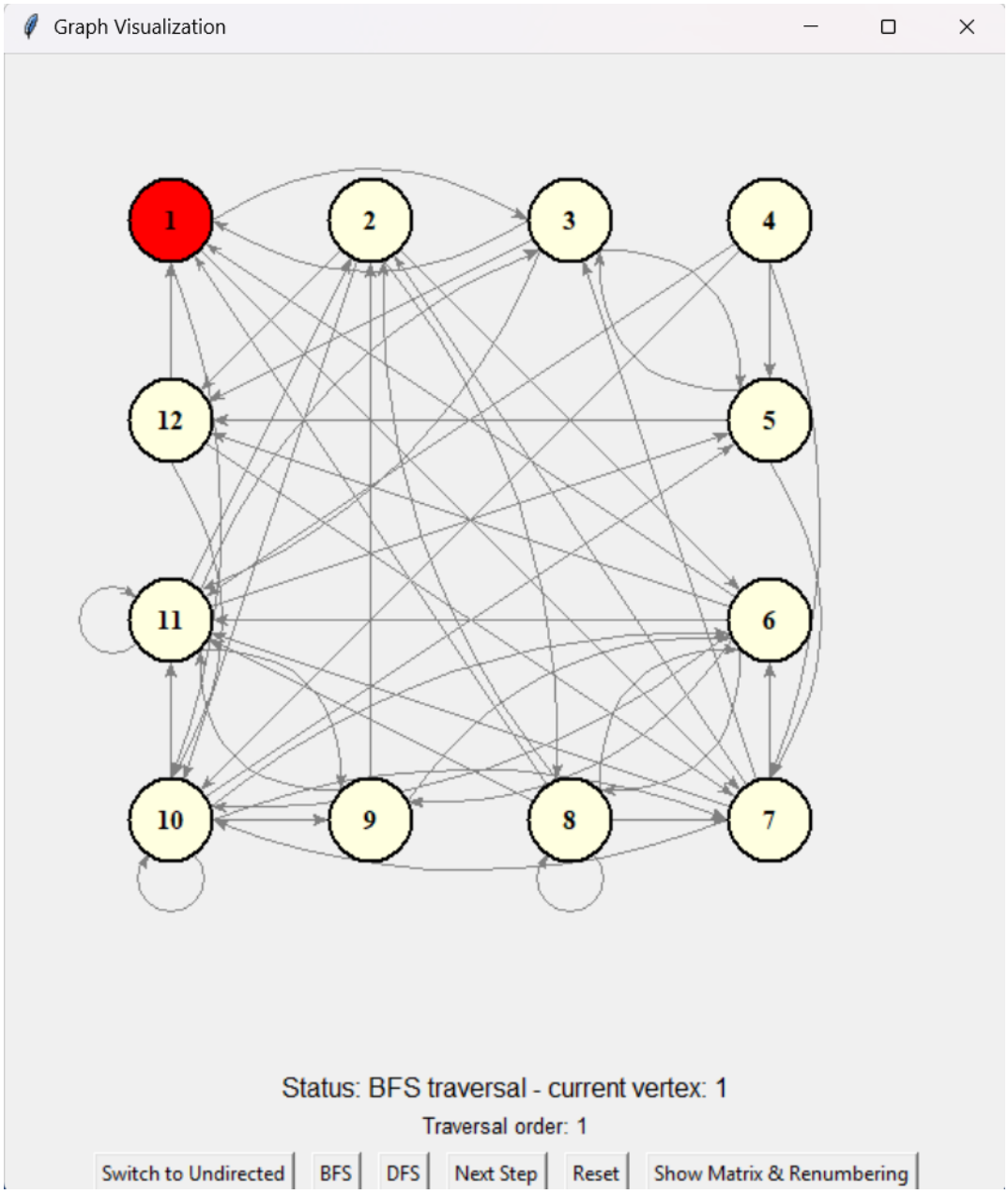
```
Directed graph:

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

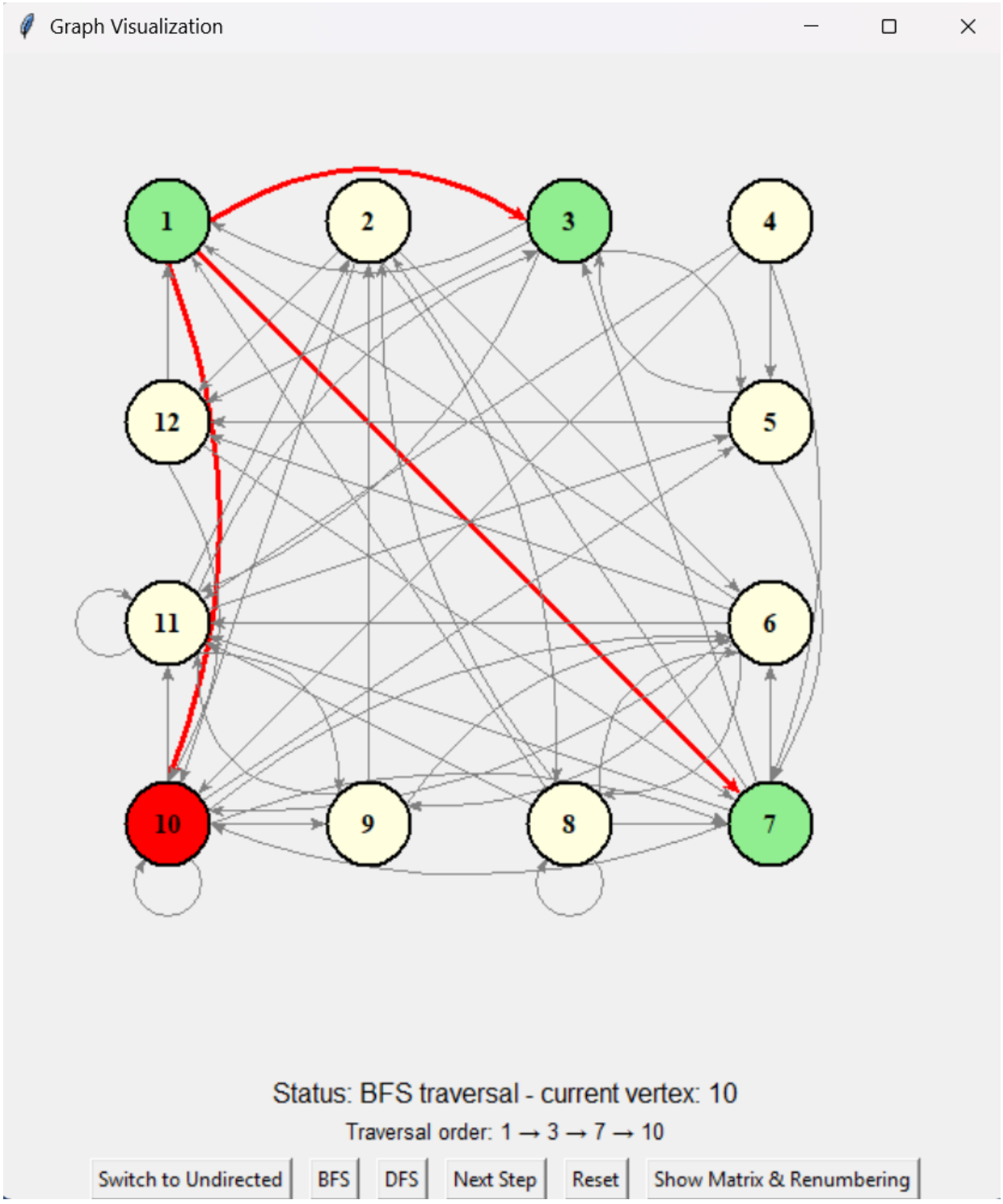
2. BFS

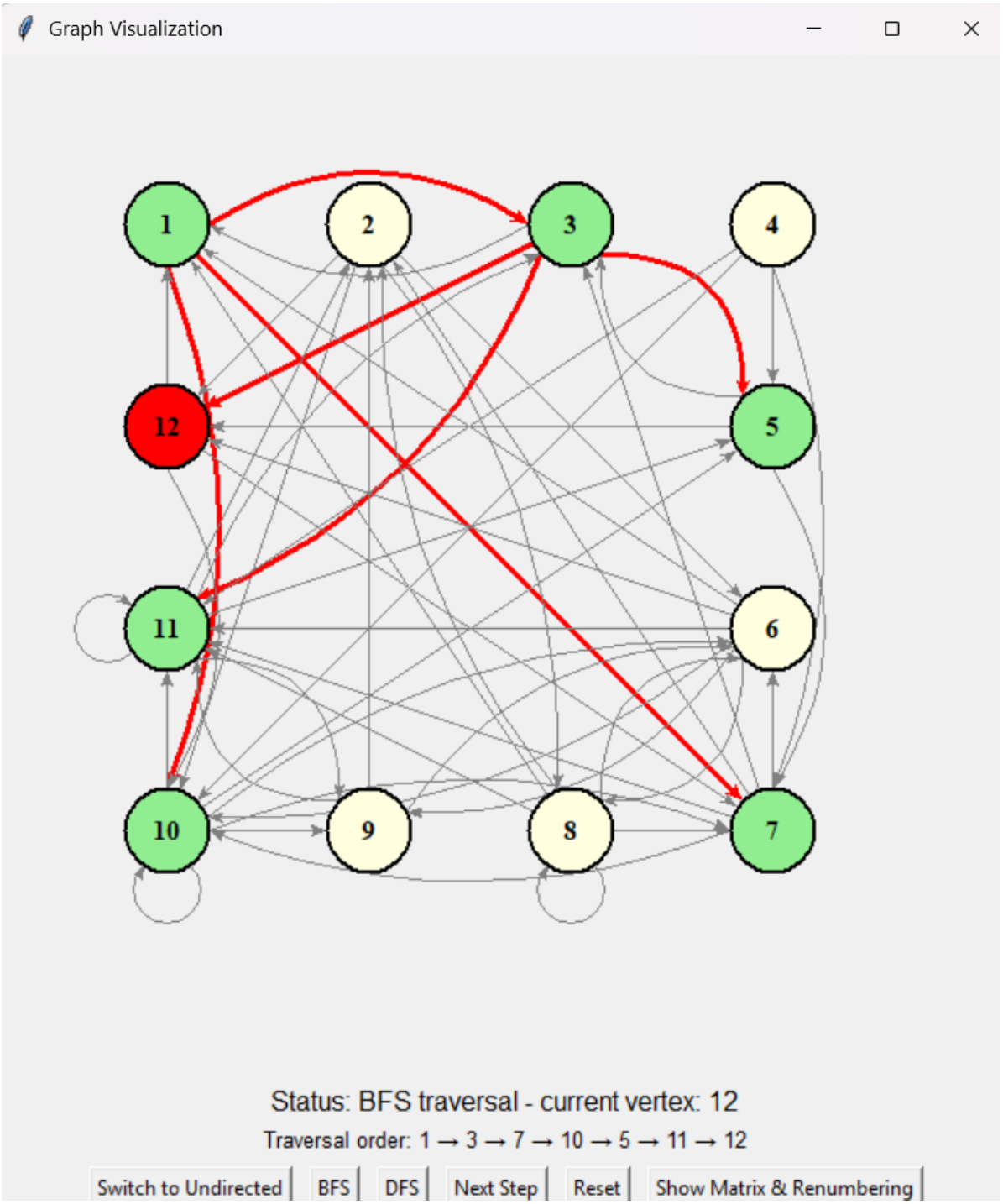
```
=== Traversal Tree Adjacency Matrix ===
    1  2  3  4  5  6  7  8  9 10 11 12
  ------------------------------------
 1| 0  0  1  0  0  0  1  0  0  1  0  0
 2| 0  0  0  0  0  0  0  1  0  0  0  0
 3| 0  0  0  0  1  0  0  0  0  0  1  1
 4| 0  0  0  0  0  0  0  0  0  0  0  0
 5| 0  0  0  0  0  0  0  0  0  0  0  0
 6| 0  0  0  0  0  0  0  0  0  0  0  0
 7| 0  1  0  0  0  1  0  0  0  0  0  0
 8| 0  0  0  0  0  0  0  0  0  0  0  0
 9| 0  0  0  0  0  0  0  0  0  0  0  0
10| 0  0  0  0  0  0  0  0  1  0  0  0
11| 0  0  0  0  0  0  0  0  0  0  0  0
12| 0  0  0  0  0  0  0  0  0  0  0  0
```

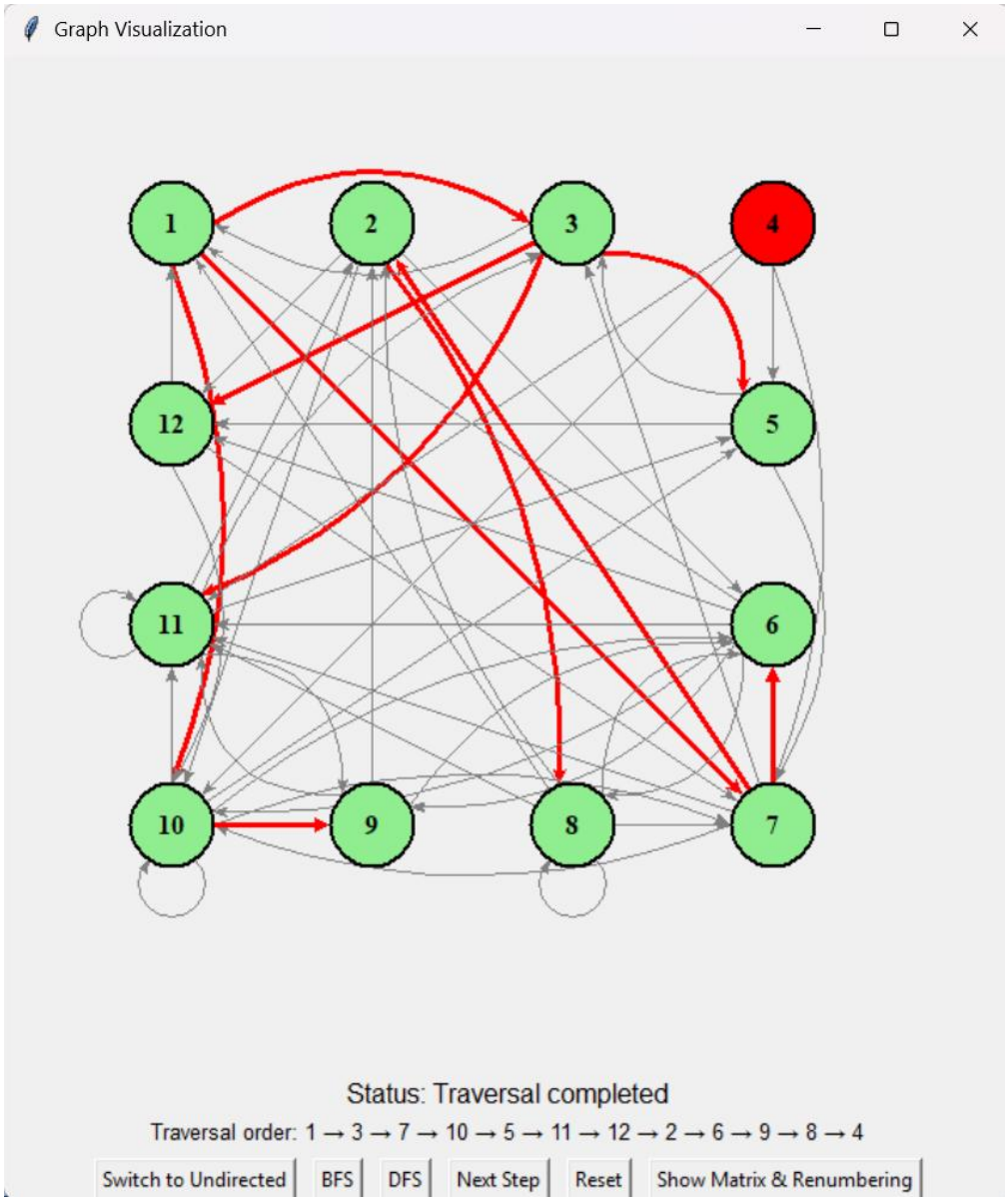
```
=== Vertex Renumbering ===
Format: [original number] -> [new number in traversal order]
 1 ->  1
 2 ->  8
 3 ->  2
 4 -> 12
 5 ->  5
 6 ->  9
 7 ->  3
 8 -> 11
 9 -> 10
10 ->  4
11 ->  6
12 ->  7
```

Етапи обходу графа:




Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: BFS traversal - current vertex: 10
Traversal order: 1 → 3 → 7 → 10
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: BFS traversal - current vertex: 12
Traversal order: 1 → 3 → 7 → 10 → 5 → 11 → 12
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering

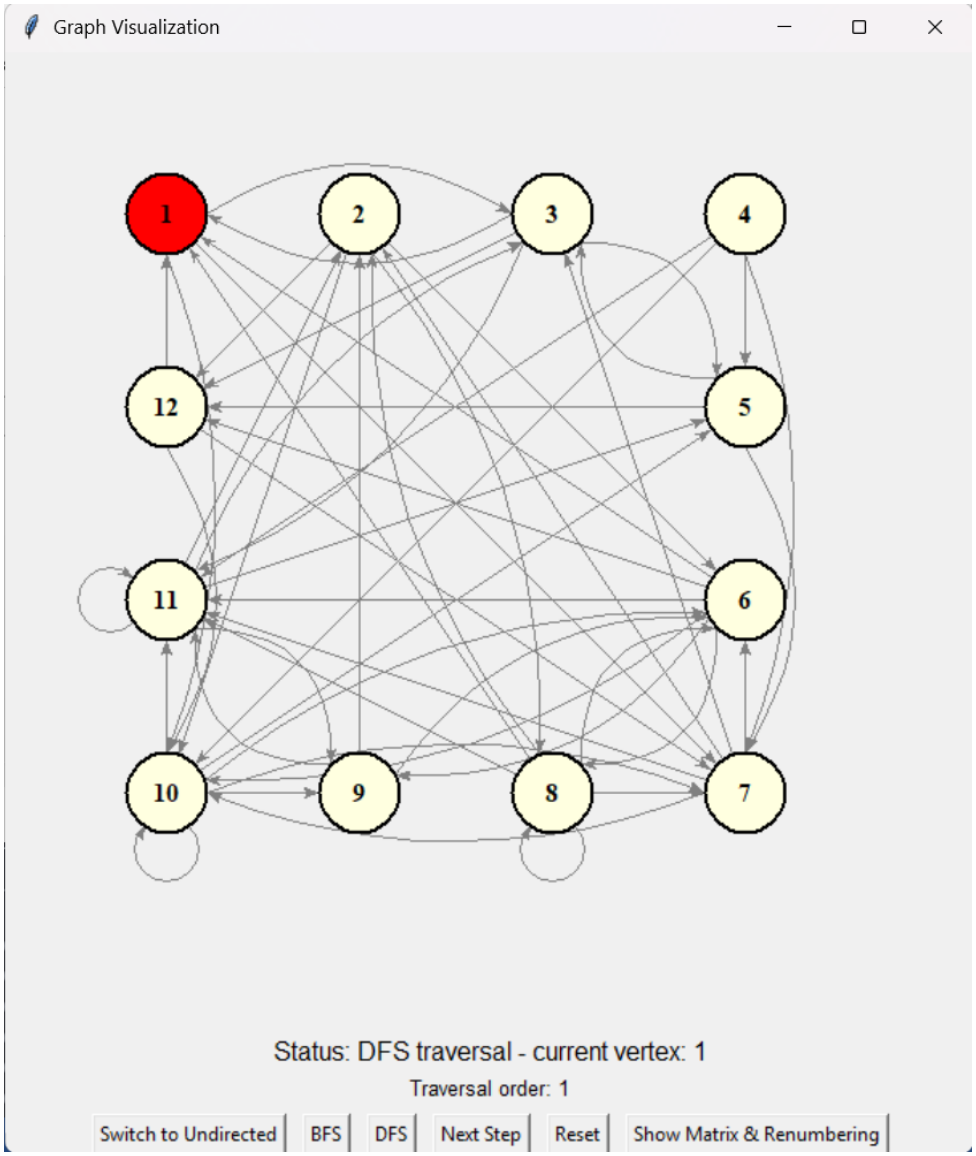


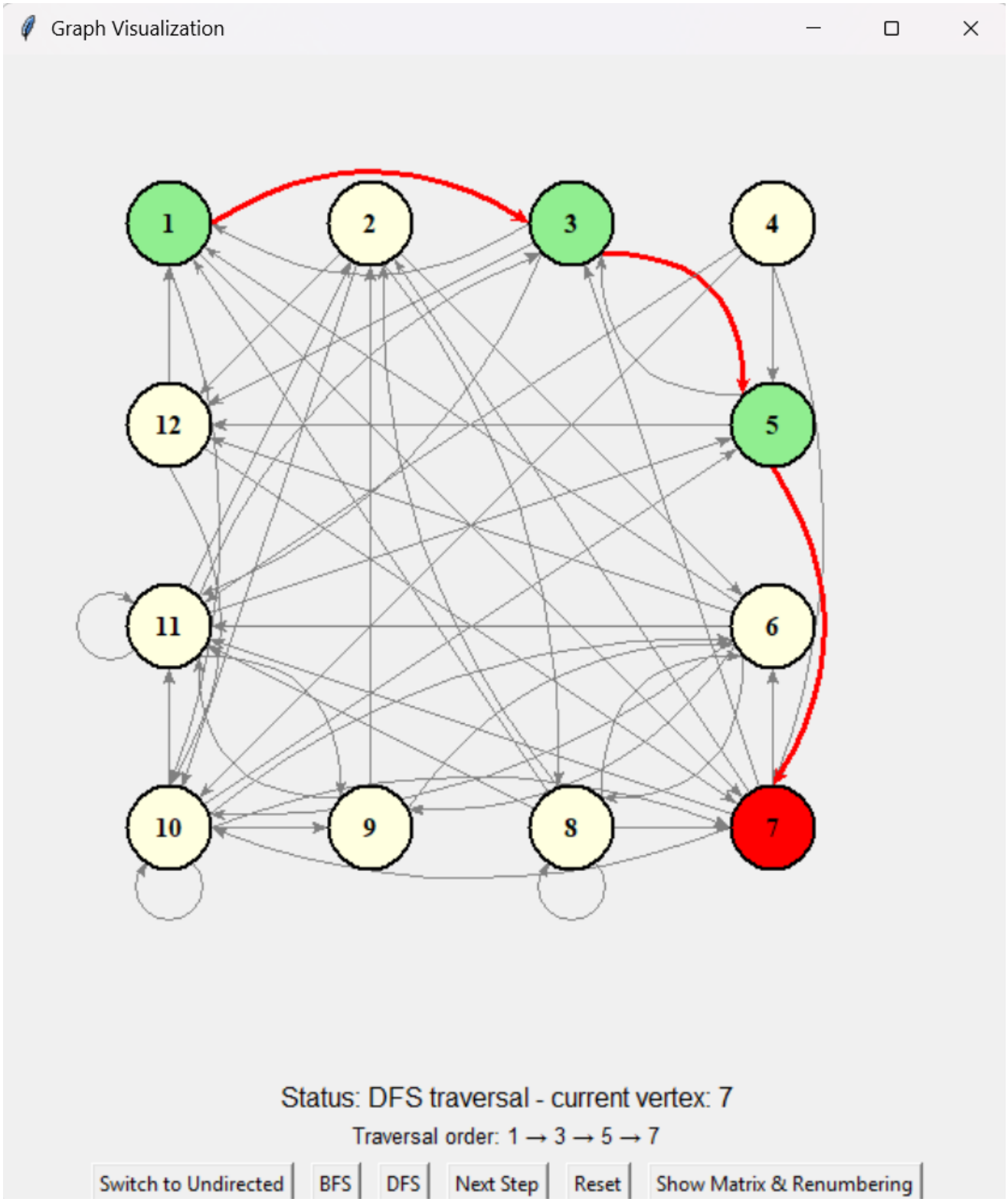
3. DFS

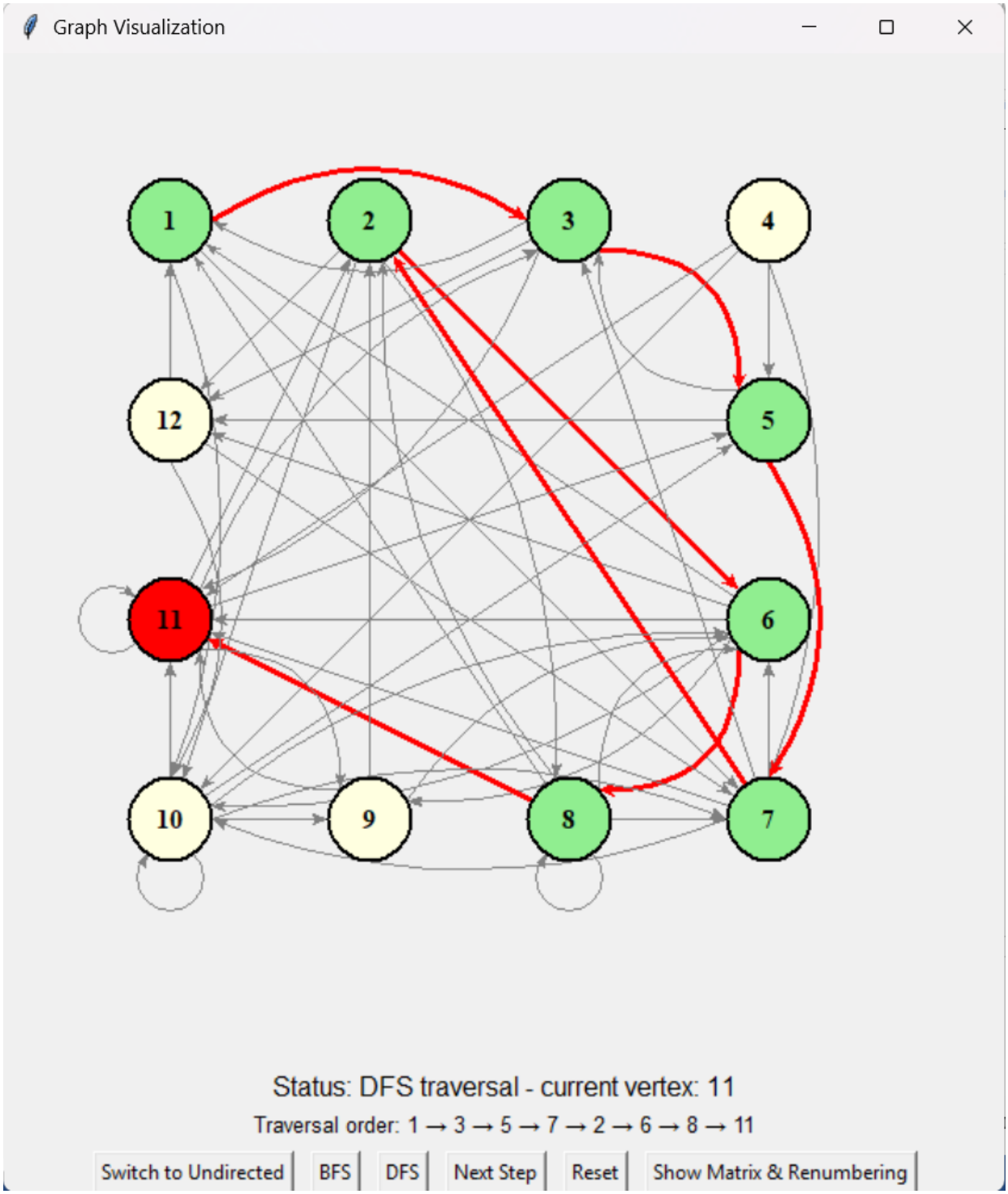
```
=== Traversal Tree Adjacency Matrix ===
    1  2  3  4  5  6  7  8  9 10 11 12
  -----------------------------------
 1| 0  0  1  0  0  0  0  0  0  0  0  0
 2| 0  0  0  0  0  1  0  0  0  0  0  0
 3| 0  0  0  0  1  0  0  0  0  0  0  0
 4| 0  0  0  0  0  0  0  0  0  0  0  0
 5| 0  0  0  0  0  0  1  0  0  0  0  0
 6| 0  0  0  0  0  0  0  1  0  1  0  1
 7| 0  1  0  0  0  0  0  0  0  0  0  0
 8| 0  0  0  0  0  0  0  0  0  0  1  0
 9| 0  0  0  0  0  0  0  0  0  0  0  0
10| 0  0  0  0  0  0  0  0  0  0  0  0
11| 0  0  0  0  0  0  0  0  1  0  0  0
12| 0  0  0  0  0  0  0  0  0  0  0  0
```

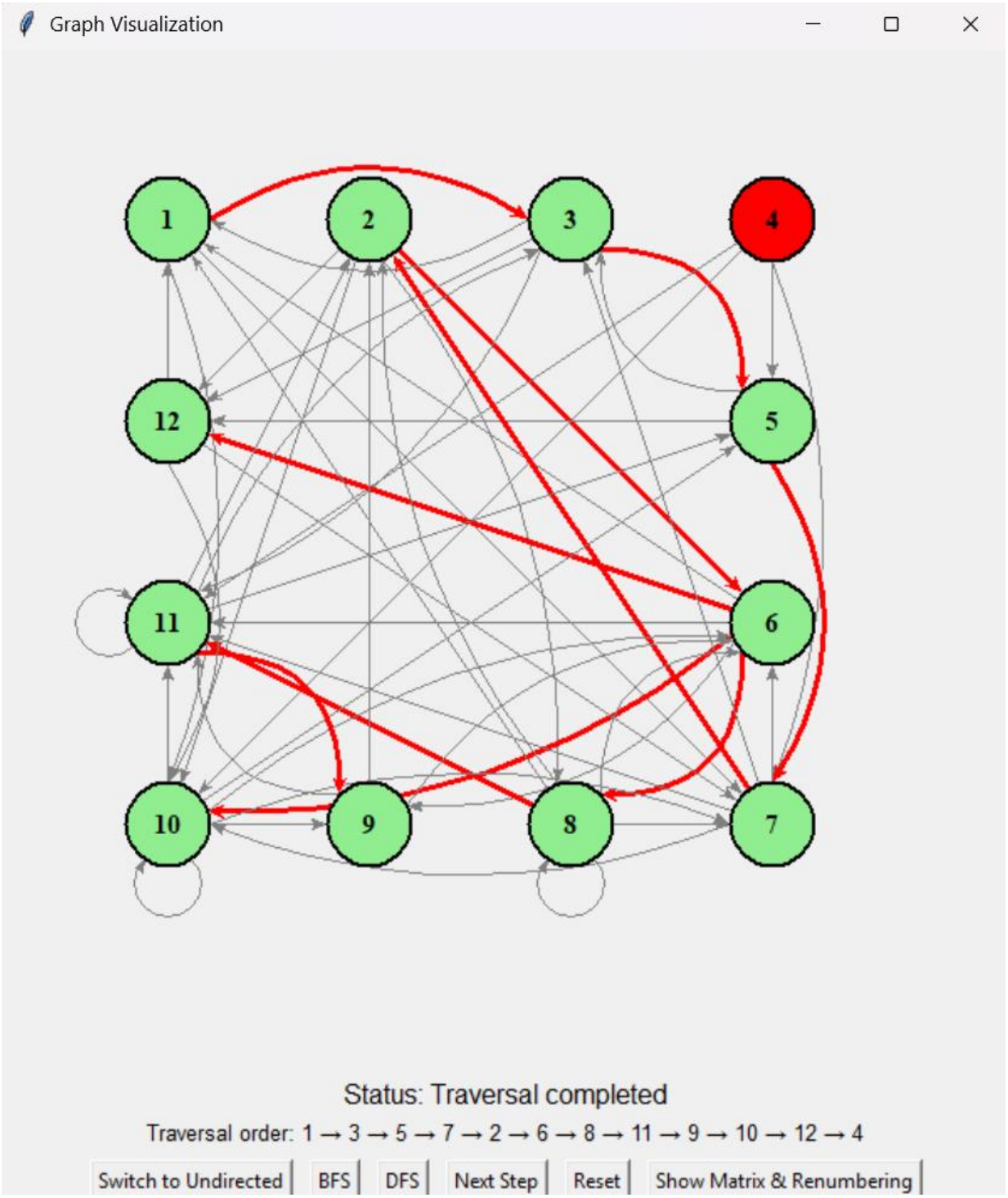
```
=== Vertex Renumbering ===
Format: [original number] -> [new number in traversal order]
 1 ->  1
 2 ->  5
 3 ->  2
 4 -> 12
 5 ->  3
 6 ->  6
 7 ->  4
 8 ->  7
 9 ->  9
10 -> 10
11 ->  8
12 -> 11
```

Етапи обходу графа:


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: DFS traversal - current vertex: 1
Traversal order: 1
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: DFS traversal - current vertex: 7
Traversal order: 1 → 3 → 5 → 7
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: DFS traversal - current vertex: 11
Traversal order: 1 → 3 → 5 → 7 → 2 → 6 → 8 → 11
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering


Graph Visualization
1
2
3
4
12
5
11
6
10
9
8
7
Status: Traversal completed
Traversal order: 1 → 3 → 5 → 7 → 2 → 6 → 8 → 11 → 9 → 10 → 12 → 4
Switch to Undirected
BFS
DFS
Next Step
Reset
Show Matrix & Renumbering

Висновки

У ході виконання роботи я закріпив навички роботи з напрямленими графами, зокрема генерування графа з заданими параметрами за допомогою генератора випадкових чисел з фіксованим seed та коефіцієнтом k. Успішно реалізував алгоритми BFS та DFS з автоматичним вибором стартової вершини (найменший номер з вихідними дугами) та покроковим виконанням. Опанував методику візуалізації обходу через виділення ребер дерева обходу кольором, динамічну зміну кольорів вершин (поточна, відвідана, невідвідана) та виведення матриці суміжності дерева обходу. Отримані результати підтвердили коректність реалізації алгоритмів та їх ефективність для аналізу різних типів графів. Ця робота значно поглибила моє розуміння алгоритмів обходу графів та методів їх візуалізації, що є важливим для подальшого вивчення складніших алгоритмів.