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 №6

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 зважений ненапрямлений граф із заданими параметрами так само, як у лабораторній роботі №3.

   Відмінність 1: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.05$.

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

   1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту $n_1 n_2 n_3 n_4$;
   2) матриця розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$;
   3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.05$, 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.

   Матриця A_{undir} ненапрямленого графа одержується з матриці A_dir так само, як у ЛР №3.

   Відмінність 2: матриця ваг W формується таким чином.

   1) матриця B розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$ (параметр генератора випадкових чисел той же самий, $n_1 n_2 n_3 n_4$);

   2) одержується матриця C:

      $c_{ij} = \text{ceil}(b_{ij} \cdot 100 \cdot a_{undir_{i,j}}), \qquad c_{i,j} \in C,\ b_{ij} \in B,\ a_{undir_{i,j}} \in A_{undir},$

      де ceil — це функція, що округляє кожен елемент матриці до найближчого цілого числа, більшого чи рівного за дане;

3) одержується матриця D, у якій

   $d_{ij} = 0$, якщо $c_{ij} = 0$,

   $d_{ij} = 1$, якщо $c_{ij} > 0$, $\quad d_{ij} \in D, c_{ij} \in C$;

4) одержується матриця H, у якій

   $h_{ij} = 1$, якщо $d_{ij} \neq d_{ji}$,

   та $h_{ij} = 0$ в іншому випадку;

5) Tr — верхня трикутна матриця з одиниць ($tr_{ij} = 1$ при $i < j$);

6) матриця ваг W симетрична, і її елементи одержуються за формулою: $w_{ij} = w_{ji} = (d_{ij} + h_{ij} \cdot tr_{ij}) \cdot c_{ij}$.

2. Створити програму для знаходження мінімального кістяка за алгоритмом Краскала при n_4 — парному і за алгоритмом Пріма — при непарному. При цьому у програмі:

   - графи представляти у вигляді динамічних списків, обхід графа, додавання, віднімання вершин, ребер виконувати як функції з вершинами відповідних списків;
   - у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3. Під час обходу графа побудувати дерево його кістяка. У програмі дерево кістяка виводити покроково у процесі виконання алгоритму. Це можна виконати одним із двох способів:

   - або виділяти іншим кольором ребра графа;
   - або будувати кістяк поряд із графом.

   При зображенні як графа, так і його кістяка, вказати ваги ребер.

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
import math

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

n = 10 + n3

random.seed(4322)

k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.05

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)
]

B = [[random.uniform(0, 2.0) for _ in range(n)] for _ in range(n)]
C = [
    [math.ceil(b  *  100  *  undirected_matrix[i][j])  for  j,  b  in
enumerate(row)]
    for i, row in enumerate(B)
]
D = [[1 if c > 0 else 0 for c in row] for row in C]
H = [[1 if D[i][j] == D[j][i] else 0 for j in range(n)] for i in range(n)]
Tr = [[1 if i < j else 0 for j in range(n)] for i in range(n)]

W = [[0 for _ in range(n)] for _ in range(n)]

for i in range(n):
```

```
    for j in range(n):
        if i == j:
            W[i][j] = 0
        else:
            weight = (D[i][j] * H[i][j] * Tr[i][j]) * C[i][j]
            W[i][j] = weight if weight != 0 else math.inf

for i in range(n):
    for j in range(i + 1, n):
        if W[i][j] != math.inf or W[j][i] != math.inf:
            W[i][j] = W[j][i] = min(
                W[i][j] if W[i][j] != math.inf else float("inf"),
                W[j][i] if W[j][i] != math.inf else float("inf"),
            )

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)),
        )


def print_weight_matrix():
    print("\n    Weight Matrix W:\n")
    header = "    " + " ".join(f"{label:>4}" for label in labels[:n])
    print(header)
    print("    " + "-" * (len(header) - 4))
    for row in range(n):
        print(
            f"{labels[row]:>2} |",
            " ".join(
                (
                    f"{W[row][column]:4d}"
                    if isinstance(W[row][column], int)
                    else ("   0" if row == column else " inf")
                )
                for column in range(n)
            ),
        )
```

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) kruskal.py – клас для створення функцій, безпосередньо пов'язаних з алгоритмом Краскала

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
```

```
    def kruskal_mst(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1 and i < len(self.graph):
            u, v, w = self.graph[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e += 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)

        return result
```

4) draw_utils.py – функції малювання 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 початку пошуку мінімального кістяка графа (start), скасування (reset) пошуку, перемикання на наступний крок (next step), показ ваг графа, суми ваг мінімального кістяка графа, кроків та ваг на цих кроках.

```
from draw_utils import *
from positions import generate_positions
from matrix_print import n, W, print_weight_matrix
from kruskal import Graph
from tkinter import Button, Tk, Canvas, Frame, Label, StringVar
import math

threshold = 1
positions = generate_positions()


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

    main_frame = Frame(graph_window)
    main_frame.pack(fill="both", expand=True, padx=10, pady=10)

    info_panel = Frame(main_frame, bg="#f0f0f0")
    info_panel.pack(side="top", fill="x", pady=(0, 10))
```

```
status_var = StringVar()
status_var.set("Press 'Start' to begin")
status_label = Label(
    info_panel,
    textvariable=status_var,
    font=("Arial", 10, "bold"),
    bg="#f0f0f0",
    padx=10,
    pady=5,
)
status_label.pack(fill="x")

weight_var = StringVar()
weight_var.set("Current weight: 0 | Total MST weight: 0")
weight_label = Label(
    info_panel,
    textvariable=weight_var,
    font=("Arial", 10),
    bg="#f0f0f0",
    padx=10,
    pady=5,
)
weight_label.pack(fill="x")

canvas = Canvas(main_frame, width=550, height=550, bg="white")
canvas.pack(side="top", fill="both", expand=True)

button_panel = Frame(main_frame)
button_panel.pack(side="bottom", fill="x", pady=(10, 0))

g = Graph(n)
for i in range(n):
    for j in range(i + 1, n):
        if W[i][j] > 0:
            g.add_edge(i, j, W[i][j])

mst_edges = g.kruskal_mst()
total_weight = sum(w for u, v, w in mst_edges)

traversal_path = []
visited = [False] * n
current_step = 0
highlighted_edges = []
current_vertex = None
next_vertex = None
mst_complete = False
```

```
    traversal_started = False

    VERTEX_RADIUS = 18
    WEIGHT_OFFSET = 35

    def calculate_weight_position(x1, y1, x2, y2, weight):
        mid_x = (x1 + x2) / 2
        mid_y = (y1 + y2) / 2

        dx = x2 - x1
        dy = y2 - y1
        length = math.hypot(dx, dy)

        if length == 0:
            return mid_x, mid_y

        perp_x = -dy / length
        perp_y = dx / length

        offset = 20
        final_x = mid_x + perp_x * offset
        final_y = mid_y + perp_y * offset

        return final_x, final_y

    def draw_edge_weight(x1, y1, x2, y2, weight, is_active=False):
        weight_x, weight_y = calculate_weight_position(x1, y1, x2, y2,
weight)

        bg_color = "#ff4444" if is_active else "white"
        fg_color = "white" if is_active else "black"
        outline = "black" if is_active else "#888888"

        canvas.create_rectangle(
            weight_x - 15,
            weight_y - 10,
            weight_x + 15,
            weight_y + 10,
            fill=bg_color,
            outline=outline,
            width=1,
        )
        canvas.create_text(
            weight_x,
            weight_y,
            text=str(weight),
```

```
            font=("Arial", 8, "bold"),
            fill=fg_color,
            tags="weight",
        )

    def build_traversal_path():
        nonlocal traversal_path, visited
        traversal_path = []
        visited = [False] * n
        stack = [0]
        visited[0] = True
        traversal_path.append(0)

        adj = [[] for _ in range(n)]
        for u, v, w in mst_edges:
            adj[u].append((v, w))
            adj[v].append((u, w))

        while stack:
            u = stack.pop()
            for v, w in sorted(adj[u], key=lambda x: x[1]):
                if not visited[v]:
                    visited[v] = True
                    traversal_path.append((u, v, w))
                    traversal_path.append(v)
                    stack.append(v)

    def draw_graph():
        nonlocal mst_complete
        canvas.delete("all")

        drawn_edges = set()
        inactive_weights = []
        active_weights = []

        for i in range(n):
            for j in range(i + 1, n):
                if W[i][j] > 0 and W[i][j] != math.inf:
                    x1, y1 = positions[i]
                    x2, y2 = positions[j]
                    weight = W[i][j]

                    edge_color = "#cccccc"
                    width = 1
                    is_active = (i, j) in highlighted_edges or (
                        j,
```

```
                        i,
                    ) in highlighted_edges

                    if is_active:
                        edge_color = "#ff4444"
                        width = 3

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

                    drawn_edges.add((min(i, j), max(i, j)))

        for i, j in drawn_edges:
            weight = max(W[i][j], W[j][i])
            if weight > 0:
                x1, y1 = positions[i]
                x2, y2 = positions[j]
                is_active = (i, j) in highlighted_edges or (j, i) in
highlighted_edges
                if is_active:
                    active_weights.append((x1, y1, x2, y2, weight, True))
                else:
                    inactive_weights.append((x1, y1, x2, y2, weight,
False))
```

```
        for x1, y1, x2, y2, weight, is_active in inactive_weights:
            draw_edge_weight(x1, y1, x2, y2, weight, is_active)

        for x1, y1, x2, y2, weight, is_active in active_weights:
            draw_edge_weight(x1, y1, x2, y2, weight, is_active)

        for i, (x, y) in enumerate(positions):
            color = "#aaccff"
            if i == current_vertex:
                color = "#ffff44"
            elif i == next_vertex:
                color = "#ffaa44"

            canvas.create_oval(
                x - VERTEX_RADIUS,
                y - VERTEX_RADIUS,
                x + VERTEX_RADIUS,
                y + VERTEX_RADIUS,
                outline="black",
                width=2,
                fill=color,
            )
            canvas.create_text(
                x, y, text=str(i + 1), font=("Arial", 10, "bold"),
fill="black"
            )

        if not traversal_started:
            status_var.set("Press 'Start' to begin")
        elif current_step == 0 and traversal_started:
            status_var.set("Starting from vertex 1")
        elif current_step < len(traversal_path):
            if isinstance(traversal_path[current_step - 1], tuple):
                u, v, w = traversal_path[current_step - 1]
                status_var.set(
                    f"Step {current_step}: Adding edge {u+1}-{v+1} (weight
{w})"
                )
            else:
                v = traversal_path[current_step - 1]
                status_var.set(f"Step {current_step}: Visiting vertex
{v+1}")
        else:
            status_var.set(f"Minimum spanning tree found! Total weight:
{total_weight}")
```

```
            mst_complete = True

        current_w = sum(
            w
            for step in traversal_path[:current_step]
            if isinstance(step, tuple)
            for u, v, w in [step]
        )
        weight_var.set(
            f"Current weight: {current_w} | Total MST weight:
{total_weight}"
        )

    def start_traversal():
        nonlocal traversal_started
        if not traversal_started:
            traversal_started = True
            build_traversal_path()
            next_step()

    def next_step():
        nonlocal current_step, current_vertex, next_vertex, mst_complete
        if not traversal_started:
            start_traversal()
            return

        if current_step < len(traversal_path):
            if isinstance(traversal_path[current_step], tuple):
                u, v, w = traversal_path[current_step]
                highlighted_edges.append((u, v))
                current_vertex = u
                next_vertex = v
            else:
                v = traversal_path[current_step]
                current_vertex = v
                next_vertex = None
            current_step += 1
        elif not mst_complete:
            mst_complete = True
        draw_graph()

    def reset():
        nonlocal current_step, highlighted_edges, current_vertex
        nonlocal next_vertex, mst_complete, traversal_started
        current_step = 0
        highlighted_edges = []
```

```
        current_vertex = None
        next_vertex = None
        mst_complete = False
        traversal_started = False
        draw_graph()

    Button(
        button_panel,
        text="Start",
        command=start_traversal,
        bg="#4CAF50",
        fg="white",
        font=("Arial", 10, "bold"),
    ).pack(side="left", padx=5, ipadx=10)
    Button(
        button_panel,
        text="Next Step",
        command=next_step,
        bg="#2196F3",
        fg="white",
        font=("Arial", 10, "bold"),
    ).pack(side="left", padx=5, ipadx=10)
    Button(
        button_panel,
        text="Reset",
        command=reset,
        bg="#f44336",
        fg="white",
        font=("Arial", 10, "bold"),
    ).pack(side="left", padx=5, ipadx=10)

    print("\n" + "=" * 50)
    print_weight_matrix()

    draw_graph()
    graph_window.mainloop()
```

6) main.py – вивід матриці суміжності ненапрямленого графа, матриці ваг графа та виконання головної графічної функції

```
from graph_draw import create_graph_window
from matrix_print import matrix_print, undirected_matrix, labels, n

matrix_print(undirected_matrix, n, labels, "Undirected graph")
```

```
create_graph_window()
```

Результати тестування програми

1. Матриця суміжності ненапрямленого графа

Undirected graph:

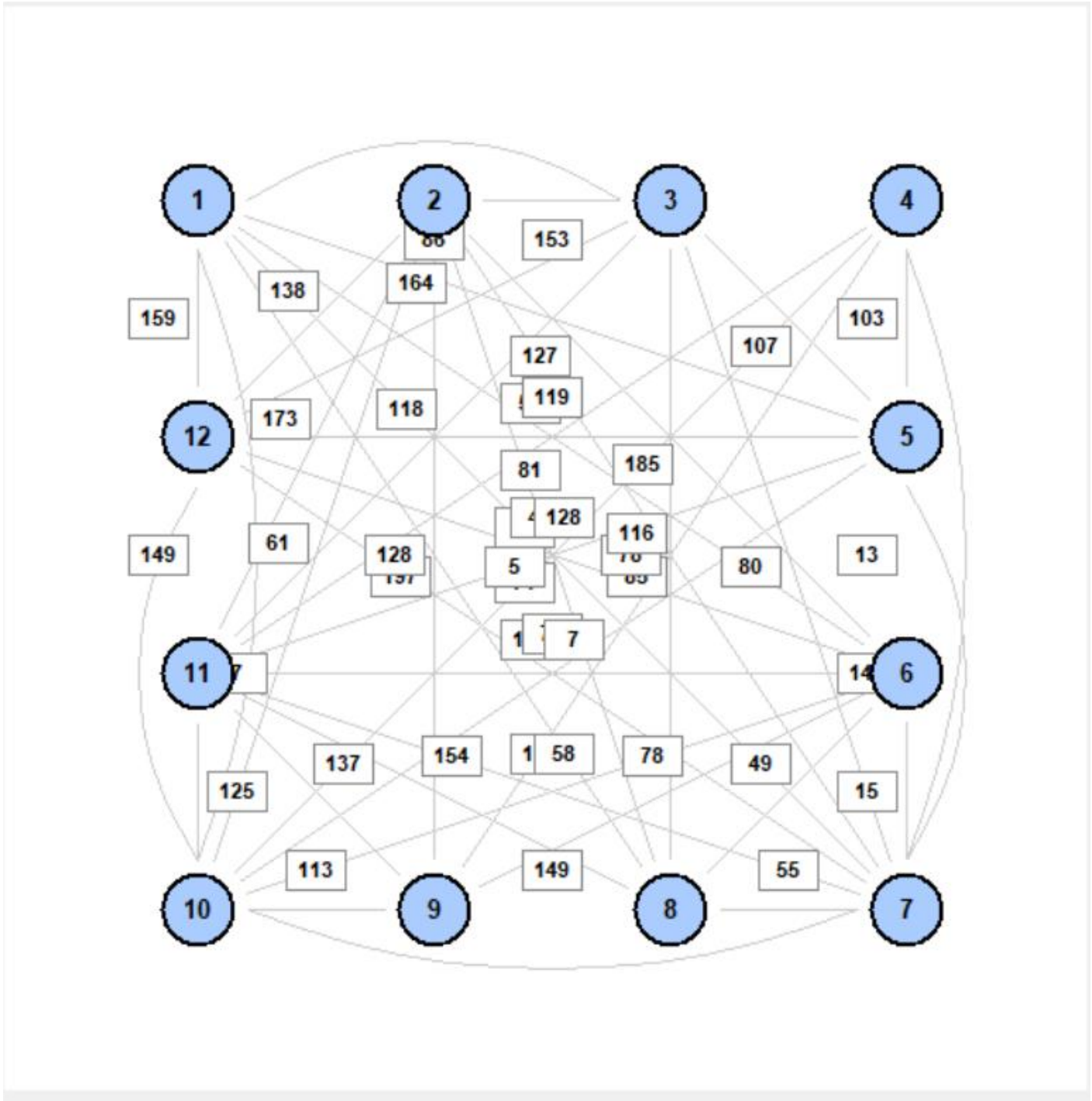
	1	2	3	4	5	6	7	8	9	A	B	C
1	1	0	1	0	1	1	1	1	0	1	0	1
2	0	0	1	0	0	1	1	1	1	1	1	1
3	1	1	0	0	1	0	1	1	0	0	1	1
4	0	0	0	0	1	0	1	0	1	1	1	0
5	1	0	1	1	0	0	1	0	0	1	1	1
6	1	1	0	0	0	0	1	1	1	1	1	1
7	1	1	1	1	1	1	0	1	0	1	1	1
8	1	1	1	0	0	1	1	1	0	0	1	0
9	0	1	0	1	0	1	0	0	0	1	1	0
A	1	1	0	1	1	1	1	0	1	1	1	1
B	0	1	1	1	1	1	1	1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

2. Матриця ваг графа

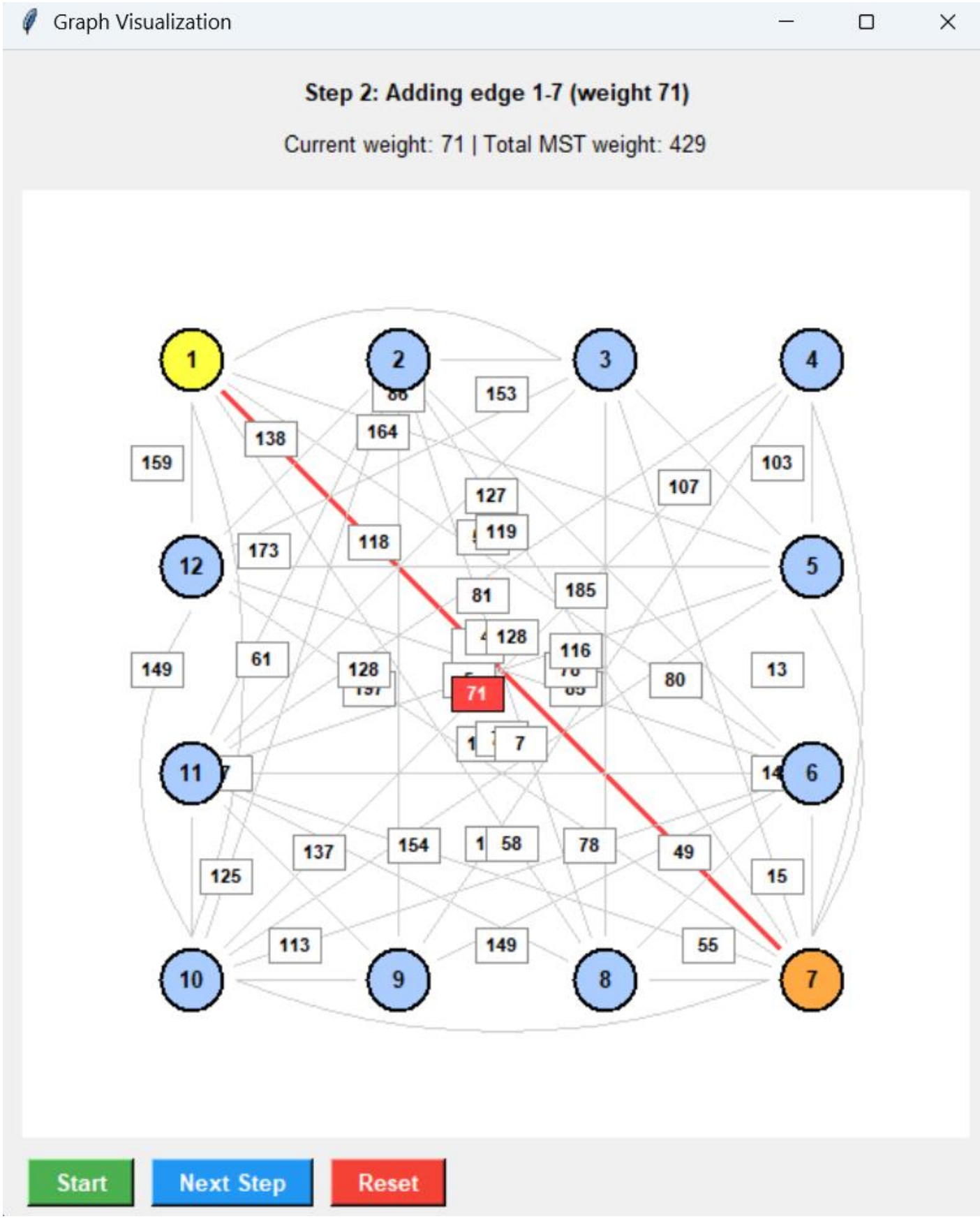
Weight Matrix W:

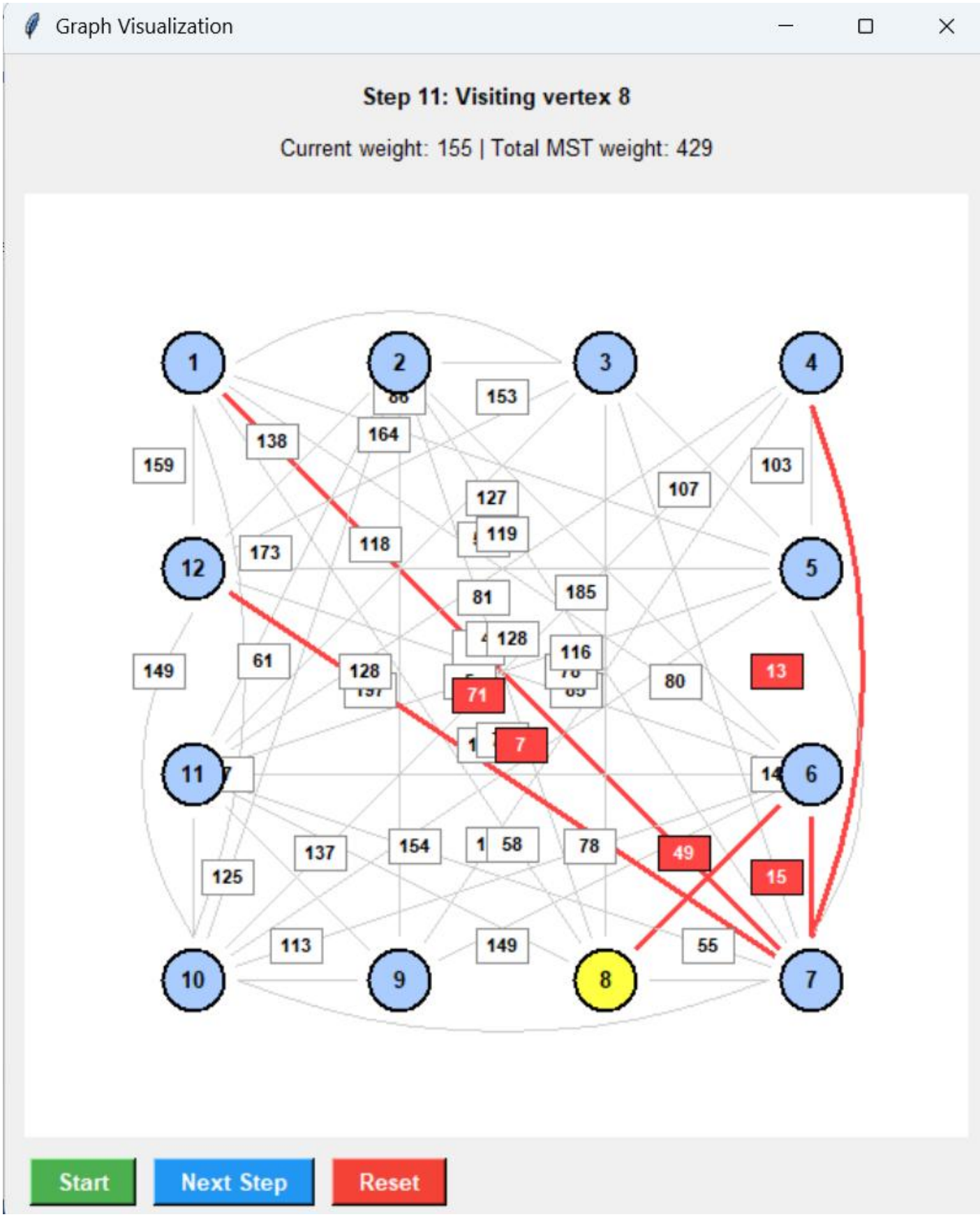
	1	2	3	4	5	6	7	8	9	A	B	C
1	0	inf	86	inf	127	81	71	197	inf	149	inf	159
2	inf	0	153	inf	inf	185	85	5	128	61	173	138
3	86	153	0	inf	107	inf	80	78	inf	inf	118	164
4	inf	inf	inf	0	103	inf	13	inf	116	93	58	inf
5	127	inf	107	103	0	inf	140	inf	inf	153	48	119
6	81	185	inf	inf	inf	0	15	49	78	106	76	128
7	71	85	80	13	140	15	0	55	inf	149	58	7
8	197	5	78	inf	inf	49	55	0	inf	inf	154	inf
9	inf	128	inf	116	inf	78	inf	inf	0	113	137	inf
A	149	61	inf	93	153	106	149	inf	113	0	125	7
B	inf	173	118	58	48	76	58	154	137	125	0	inf
C	159	138	164	inf	119	128	7	inf	inf	7	inf	0

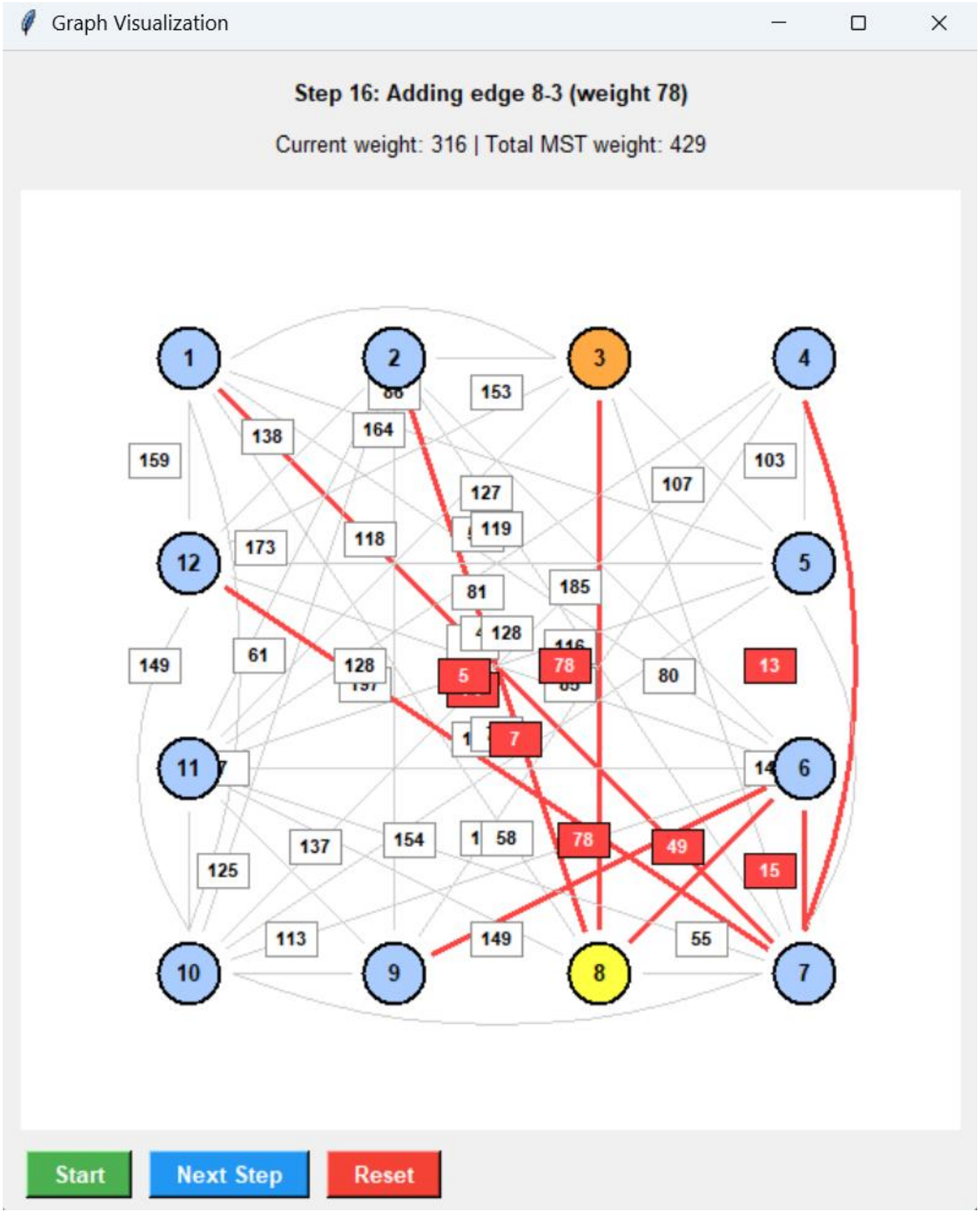
3. Граф

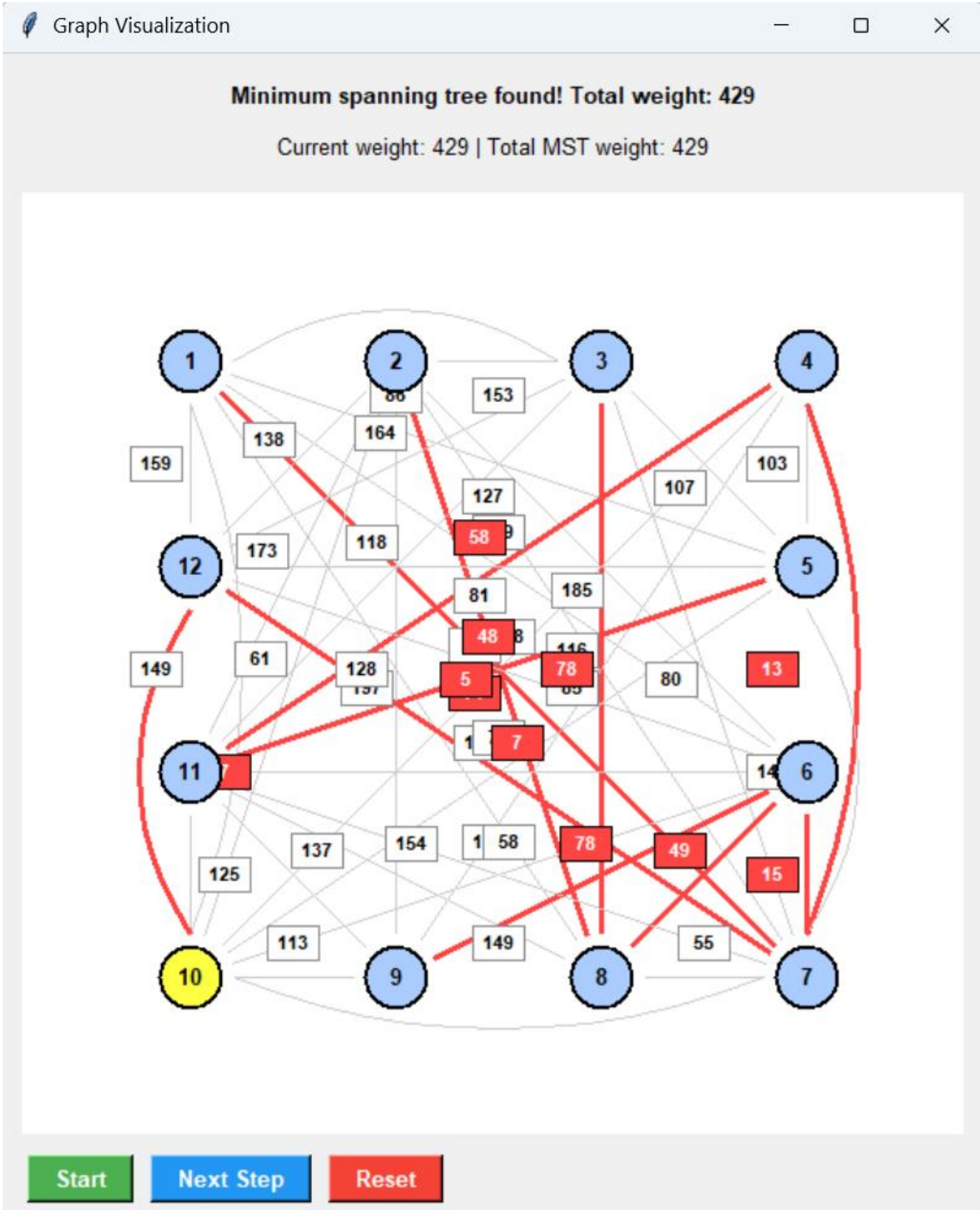


4. Пошук мінімального кістяка графа


Graph Visualization
Step 2: Adding edge 1-7 (weight 71)
Current weight: 71 | Total MST weight: 429
1
2
3
4
5
6
7
8
9
10
11
12
153
164
138
159
103
107
127
119
118
173
81
185
128
116
61
149
128
80
13
71
7
137
154
58
78
49
125
15
113
149
55
Start
Next Step
Reset


Graph Visualization
Step 11: Visiting vertex 8
Current weight: 155 | Total MST weight: 429
1
2
3
4
5
6
7
8
9
10
11
12
153
164
138
159
103
107
127
119
118
173
81
185
128
116
61
149
128
80
13
71
7
137
154
58
78
49
15
125
113
149
55
Start
Next Step
Reset


Graph Visualization
Step 16: Adding edge 8-3 (weight 78)
Current weight: 316 | Total MST weight: 429
1
2
3
4
5
6
7
8
9
10
11
12
153
164
138
159
103
107
127
119
118
173
81
185
128
61
149
80
13
78
5
7
137
154
58
49
15
125
113
149
55
Start
Next Step
Reset


Graph Visualization
Minimum spanning tree found! Total weight: 429
Current weight: 429 | Total MST weight: 429
1
2
3
4
5
6
7
8
9
10
11
12
153
138
164
159
103
107
127
58
173
118
81
185
48
61
149
128
78
80
13
7
137
154
58
78
49
125
15
113
149
55
Start
Next Step
Reset

Висновки

У ході виконання роботи я навчився працювати зі зваженими графами, зокрема освоїв методи їх генерації з урахуванням заданих параметрів. Зумів знайти та вивести в консоль матрицю ваг графа. Я реалізував алгоритм Краскала для знаходження мінімального кістяка, що дозволило мені краще зрозуміти принципи роботи з графами. Реалізував інтерактивний графічний інтерфейс з можливістю покрокового виконання алгоритму, що показував кроки, ваги та суму ваг графа. Робота дала мені цінний досвід у створенні програм для аналізу графів. Отримані знання стануть в пригоді при вивченні складніших алгоритмів та структур даних у майбутньому.