**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: Перевірив:

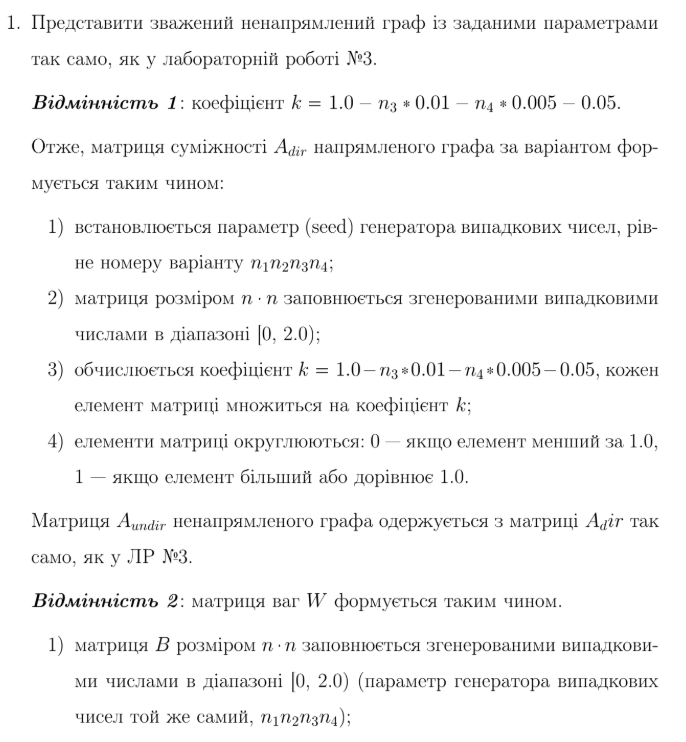
студент групи IM-43 Сергієнко А. М.

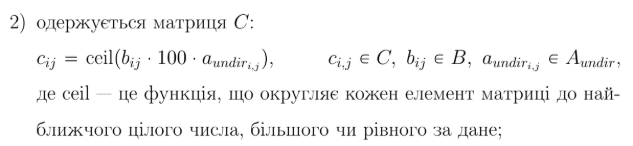
Олексійчук Станіслав Юрійович

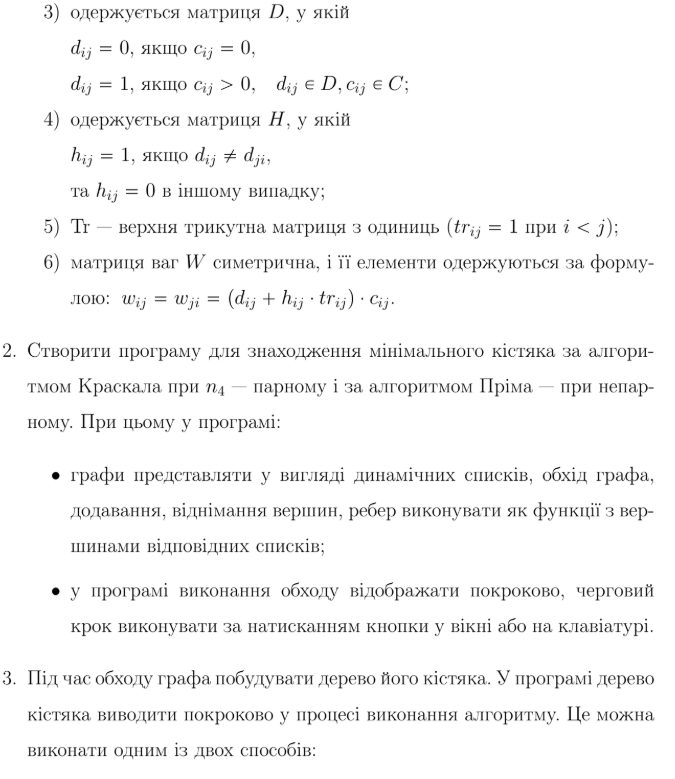
номер у списку групи: 22

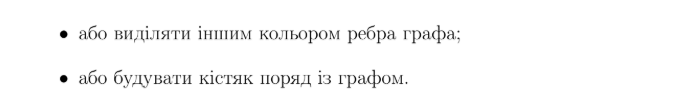
Київ 2025

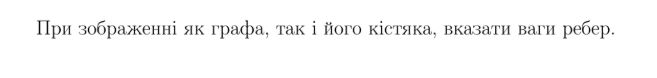
***Постановка задачі***











***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)

),

)

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. 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

1. 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

1. 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()

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

from graph\_draw import create\_graph\_window

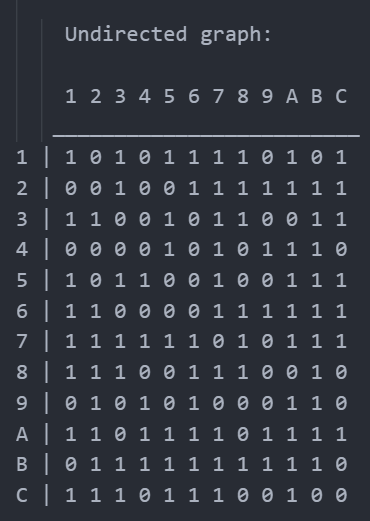
from matrix\_print import matrix\_print, undirected\_matrix, labels, n

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

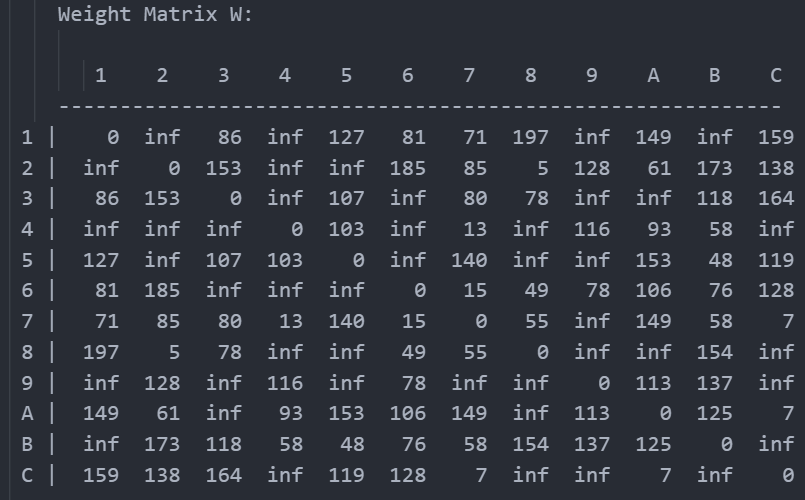
create\_graph\_window()

***Результати тестування програми***

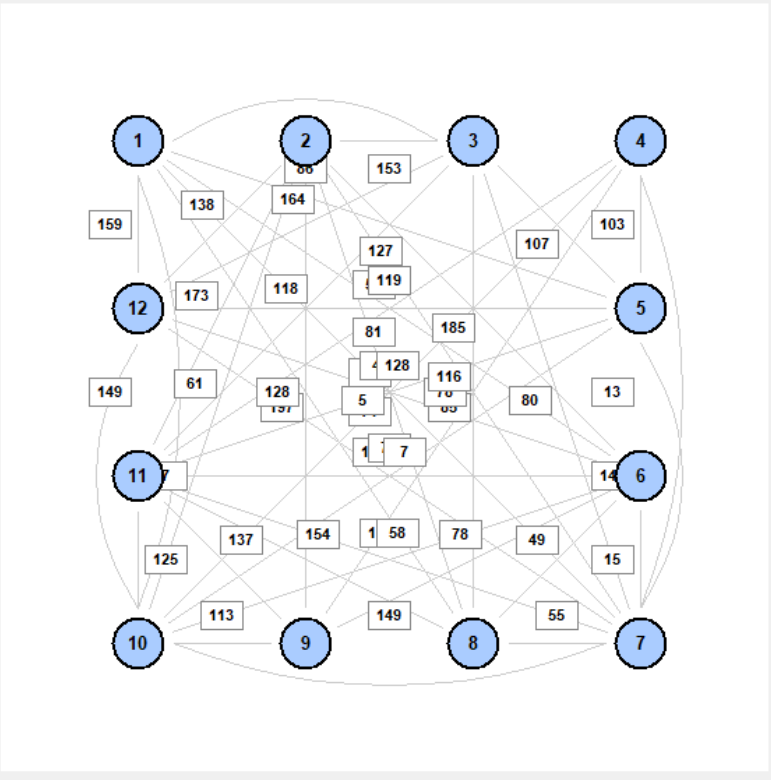
1. Матриця суміжності ненапрямленого графа



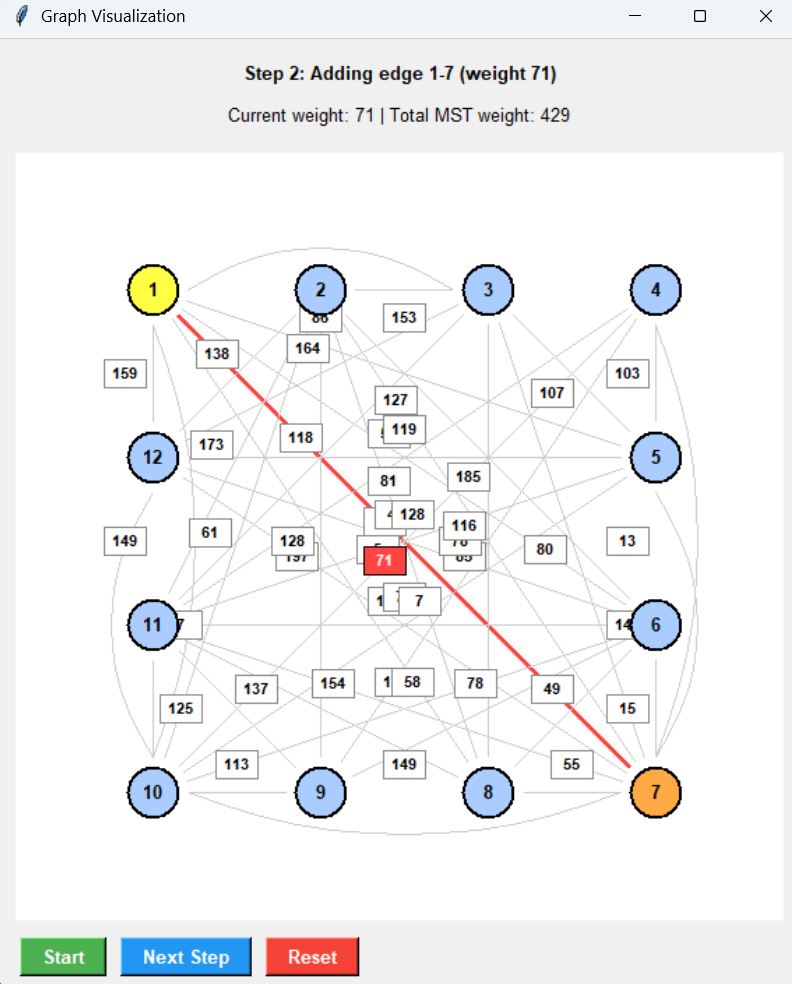
1. Матриця ваг графа

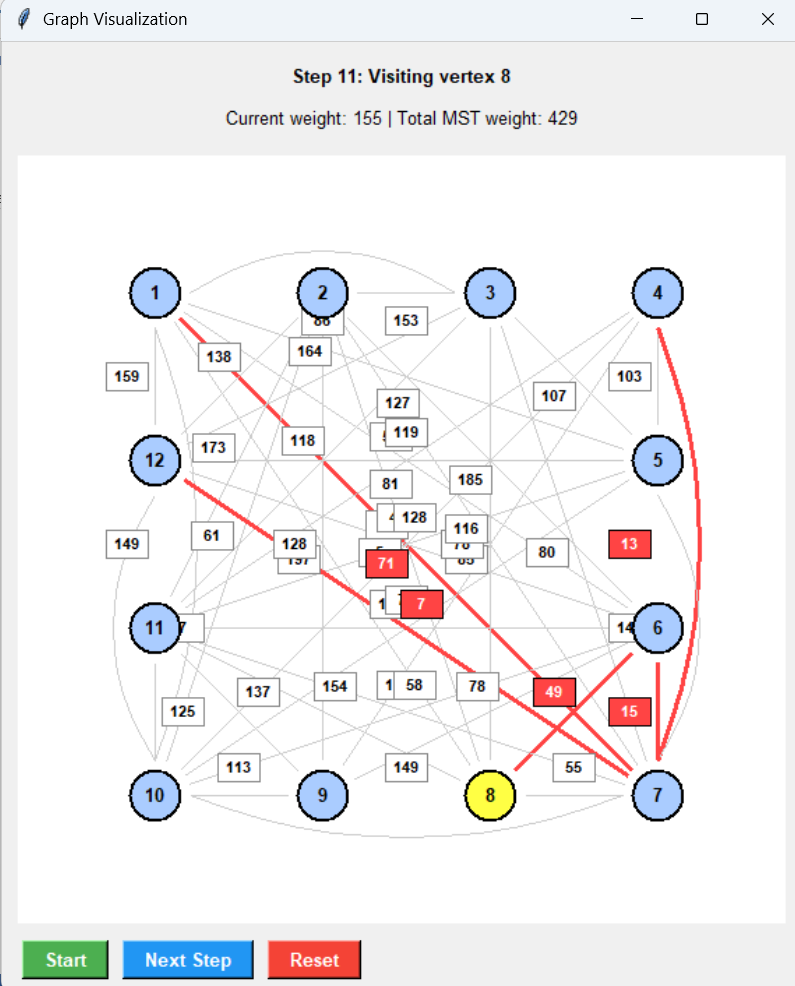


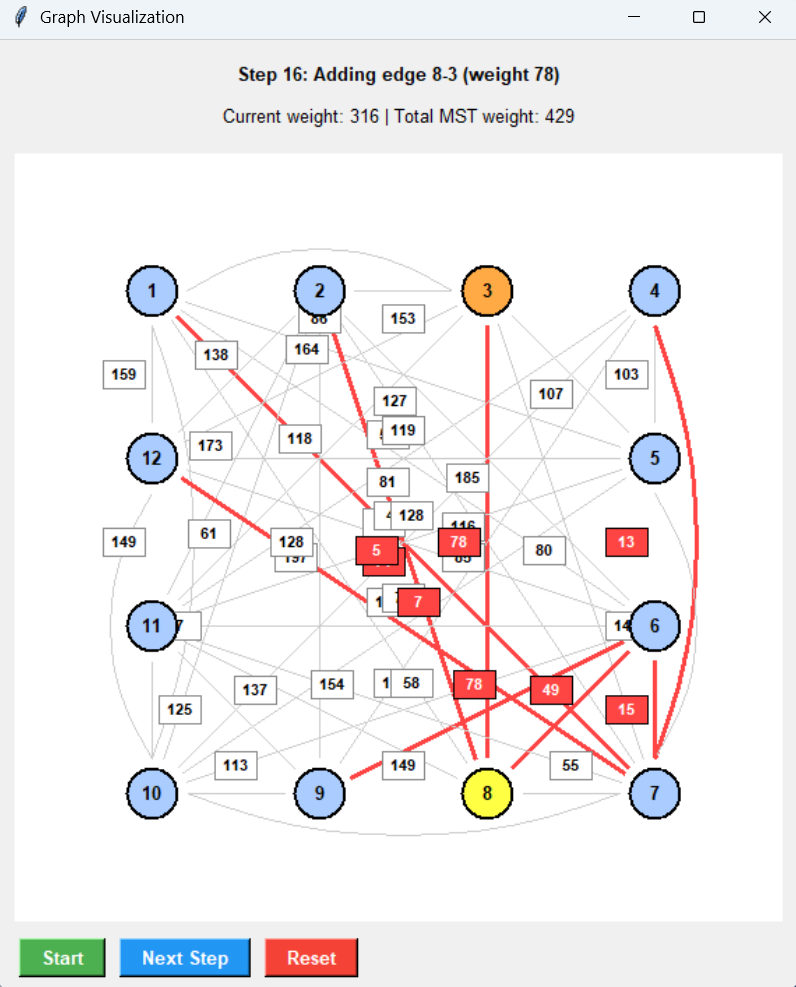
1. Граф

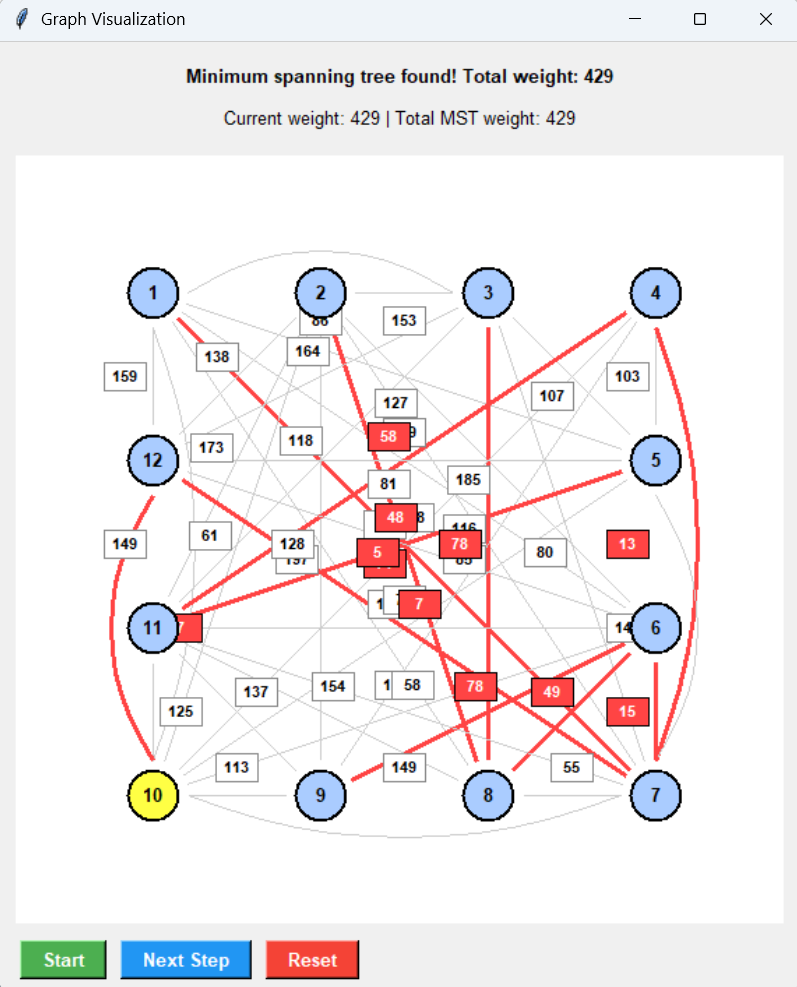


1. Пошук мінімального кістяка графа









***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.