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-42
Федоренко Іван Русланович
номер у списку групи: 29

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 - n3 * 0.01 - n4 * 0.005 - 0.05.

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

1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту n1n2n3n4;

2) матриця розміром n * n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт k, 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 Aundir ненапрямленого графа одержується з матриці Adir так само, як у ЛР №3.

Відмінність 2:

матриця ваг W формується таким чином.

1) матриця B розміром n * n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0) (параметр генератора випадкових чисел той же самий, n1n2n3n4);

2) одержується матриця C:

$$c_{ij} = ceil\left(b_{ij} * 100 * aund_{ij}\right)$$

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

3) одержується матриця D, у якій

$$d_{ij} = 0, \text{якщо } c_{ij} = 0$$
$$d_{ij} = 1, \text{якщо } c_{ij} > 0$$

4) одержується матриця H, у якій

$$h_{ij} = 1, \text{якщо } d_{ij} = d_{ji}$$
$$h_{ij} = 0, \text{у іншому випадку}$$

5) Tr — верхня трикутна матриця з одиниць ($tr_{ij} = 0, \text{якщо } i < j$);

6) матриця ваг W симетрична, і її елементи одержуються за формулою:

$$w_{ij} = w_{ji} = d_{ij} * h_{ij} * tr_{ij} * c_{ij}$$

2. Створити програму для знаходження мінімального кістяка за алгоритмом Краскала при n4 — 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.

n1n2n3n4: 4229
Варіант: 4229.
Кількість вершин = 12.
Розміщення: прямокутником з вершиною в центрі.
Алгоритм Пріма, бо n4=9, є непарним.

Програмний код:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button

class Vertex:
    def __init__(self, id, x=0, y=0):
        self.id = id
        self.x = x
        self.y = y
        self.adjacency = []

class Graph:
    def __init__(self):
        self.vertices = {}

    def add_vertex(self, id, x=0, y=0):
        self.vertices[id] = Vertex(id, x, y)

    def add_edge(self, u, v, weight):
        self.vertices[u].adjacency.append((v, weight))
        self.vertices[v].adjacency.append((u, weight))

def generate_adjacency_matrix(n, variant_number, n3, n4):
    np.random.seed(variant_number)
    T = np.random.random((n, n)) * 2.0
    k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.05
    A = (T * k >= 1.0).astype(int)
    return A

def get_undirected_matrix(A):
    return np.maximum(A, A.T)

def generate_weight_matrix(Aundir, B):
    C = np.ceil(B * 100 * Aundir).astype(int)
    D = (C > 0).astype(int)
    H = (D == D.T).astype(int)
    Tr = np.triu(np.ones_like(Aundir), k=0)

    W = np.zeros_like(C, dtype=int)
    n = C.shape[0]
    for i in range(n):
        for j in range(n):
            if i <= j:
                W[i, j] = D[i, j] * H[i, j] * Tr[i, j] * C[i, j]
            else:
                W[i, j] = W[j, i]

    return W
```

```
def get_vertex_positions(n, n4):
    positions = np.zeros((n, 2))
    if n4 in [8, 9]:
        width, height = 12, 8
        perimeter_vertices = n - 1
        corners = 4
        sides = [0, 0, 0, 0]

        for i in range(perimeter_vertices - corners):
            sides[i % 4] += 1

        idx = 0
        positions[idx] = [-width / 2, height / 2]; idx += 1

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

        positions[idx] = [width / 2, height / 2]; idx += 1

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

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

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

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

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

        positions[n - 1] = [0, 0]

    return positions

def build_graph(Aundir, W, positions):
    graph = Graph()
    n = Aundir.shape[0]
    for i in range(n):
        graph.add_vertex(i + 1, positions[i][0], positions[i][1])
    for i in range(n):
        for j in range(i + 1, n):
            if Aundir[i][j] and W[i][j] > 0:
```

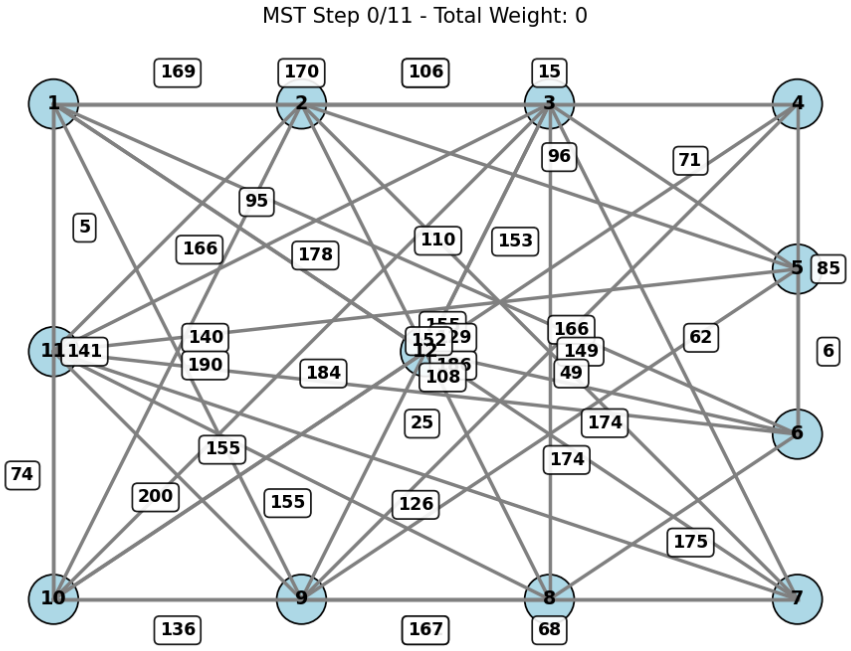
```
                graph.add_edge(i + 1, j + 1, W[i][j])
    return graph

def generate_prim_steps(graph, start=1):
    steps = []
    visited = {start}
    mst = []
    steps.append([])
    vertices = graph.vertices

    while len(visited) < len(vertices):
        edges = []
        for u in visited:
            for v, w in vertices[u].adjacency:
                if v not in visited:
                    edges.append((w, u, v))
        if not edges:
            break
        edges.sort()
        w, u, v = edges[0]
        visited.add(v)
        mst.append((u, v, w))
        steps.append(mst.copy())
    return steps

class StepVisualizer:
    def __init__(self, graph, steps):
        self.graph = graph
        self.steps = steps
        self.index = 0

        self.fig, self.ax = plt.subplots(figsize=(12, 10))
        plt.subplots_adjust(bottom=0.2)
        axprev = plt.axes([0.25, 0.05, 0.2, 0.075])
        axnext = plt.axes([0.55, 0.05, 0.2, 0.075])
        self.btn_prev = Button(axprev, 'Previous')
        self.btn_next = Button(axnext, 'Next')
        self.btn_prev.on_clicked(self.prev)
        self.btn_next.on_clicked(self.next)
        self.draw()

    def draw(self):
        self.ax.clear()
        mst_edges = set((min(u, v), max(u, v)) for u, v, _ in self.steps[self.index])
        total_weight = sum(w for _, _, w in self.steps[self.index])

        for u in self.graph.vertices:
            for v, w in self.graph.vertices[u].adjacency:
                if u < v:
```

```
                    x1, y1 = self.graph.vertices[u].x, self.graph.vertices[u].y
                    x2, y2 = self.graph.vertices[v].x, self.graph.vertices[v].y
                    color = 'red' if (u, v) in mst_edges else 'gray'
                    self.ax.plot([x1, x2], [y1, y2], color=color, linewidth=2)

                    mx, my = (x1 + x2)/2, (y1 + y2)/2
                    dx, dy = x2 - x1, y2 - y1
                    length = np.sqrt(dx**2 + dy**2)

                    offset_x = -0.5 * dy / (length + 1e-9)
                    offset_y = 0.5 * dx / (length + 1e-9)

                    self.ax.text(mx + offset_x, my + offset_y, str(w),
                                 fontsize=10, fontweight='bold',
                                 ha='center', va='center',
                                 bbox=dict(facecolor='white', edgecolor='black',
                                           boxstyle='round,pad=0.3', alpha=0.9))

        for v in self.graph.vertices.values():
            self.ax.add_patch(plt.Circle((v.x, v.y), 0.4, color='lightblue', ec='black'))
            self.ax.text(v.x, v.y, str(v.id), ha='center', va='center', fontsize=11,
fontweight='bold')

        self.ax.set_title(f"MST Step {self.index}/{len(self.steps)-1} - Total Weight:
{total_weight}")
        self.ax.axis('equal')
        self.ax.axis('off')
        self.fig.canvas.draw_idle()

    def next(self, event=None):
        if self.index < len(self.steps) - 1:
            self.index += 1
            self.draw()

    def prev(self, event=None):
        if self.index > 0:
            self.index -= 1
            self.draw()

if __name__ == '__main__':
    variant = 4229
    n1, n2, n3, n4 = 4, 2, 2, 9
    n = 10 + n3

    A = generate_adjacency_matrix(n, variant, n3, n4)
    Aundir = get_undirected_matrix(A)
    np.random.seed(variant)
    B = np.random.random((n, n)) * 2.0
    W = generate_weight_matrix(Aundir, B)
```

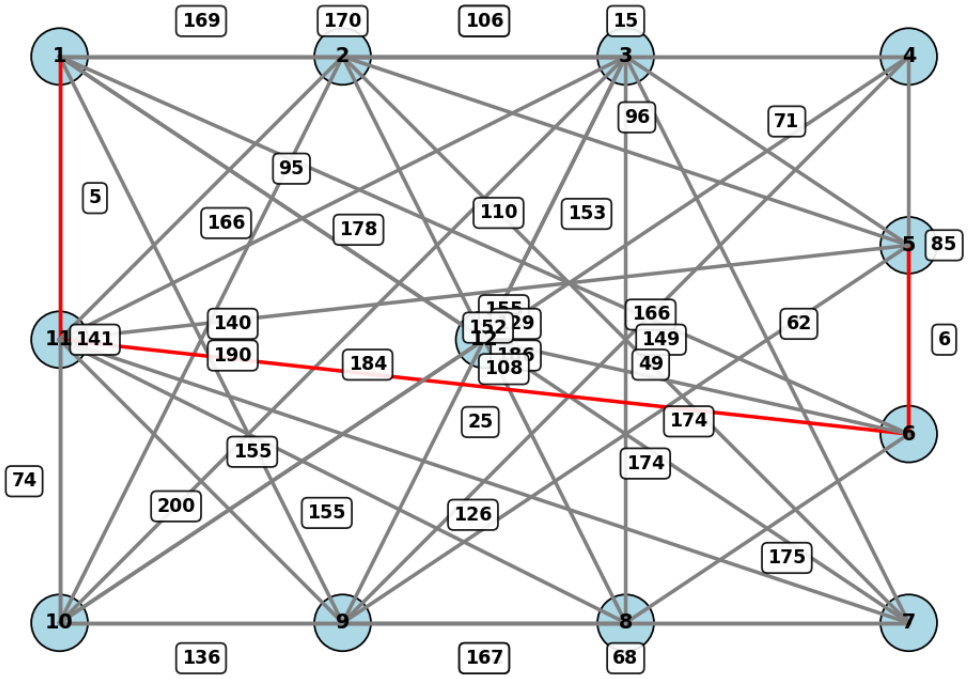
```
print("Adjacency Matrix (Undirected):")
print(Aundir)
print("\nWeight Matrix:")
print(W)

positions = get_vertex_positions(n, n4)
graph = build_graph(Aundir, W, positions)

steps = generate_prim_steps(graph, start=1)
StepVisualizer(graph, steps)
plt.show()
```

Вивід програми:

```
Adjacency Matrix (Undirected):
[[1 1 1 1 0 1 1 0 1 1 1 1]
 [1 1 1 1 1 0 1 1 0 1 1 0]
 [1 1 0 0 1 0 1 1 1 1 1 1]
 [1 1 0 0 0 1 0 0 1 1 0 0]
 [0 1 1 0 0 1 0 0 1 0 1 0]
 [1 0 0 1 1 0 0 1 0 0 1 1]
 [1 1 1 0 0 0 0 0 1 1 1 0]
 [0 1 1 0 0 1 0 0 1 0 1 0]
 [1 0 1 1 1 0 1 1 1 1 1 0]
 [1 1 1 1 0 0 1 0 1 1 1 1]
 [1 1 1 0 1 1 1 1 1 1 1 0]
 [1 0 1 0 0 1 0 0 0 1 0 0]]

Weight Matrix:
[[134 169 170 184   0 110 155   0 140 141   5  95]
 [169 149 106  15  96   0 166 129   0 190 166   0]
 [170 106   0   0  71   0  62 149 186 184 178 153]
 [184  15   0   0   0  85   0   0  49 108   0   0]
 [  0  96  71   0   0   6   0   0 174   0 152   0]
 [110   0   0  85   6   0   0 175   0   0  25 174]
 [155 166  62   0   0   0   0   0  68 192 126   0]
 [  0 129 149   0   0 175   0   0 167   0 155   0]
 [140   0 186  49 174   0  68 167 186 136 200   0]
 [141 190 184 108   0   0 192   0 136 192  74 155]
 [  5 166 178   0 152  25 126 155 200  74 135   0]
 [ 95   0 153   0   0 174   0   0   0 155   0   0]]
```



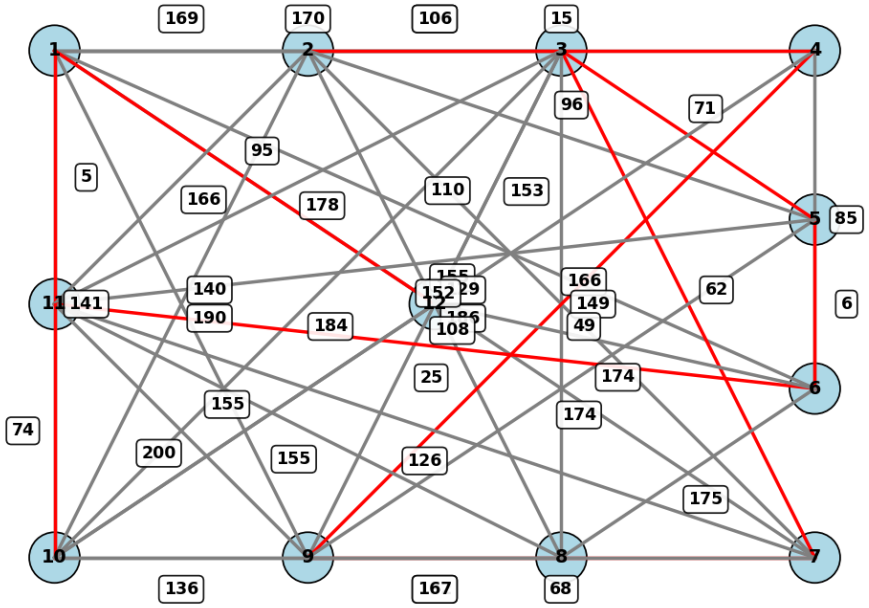
MST Step 3/11 - Total Weight: 36



Previous

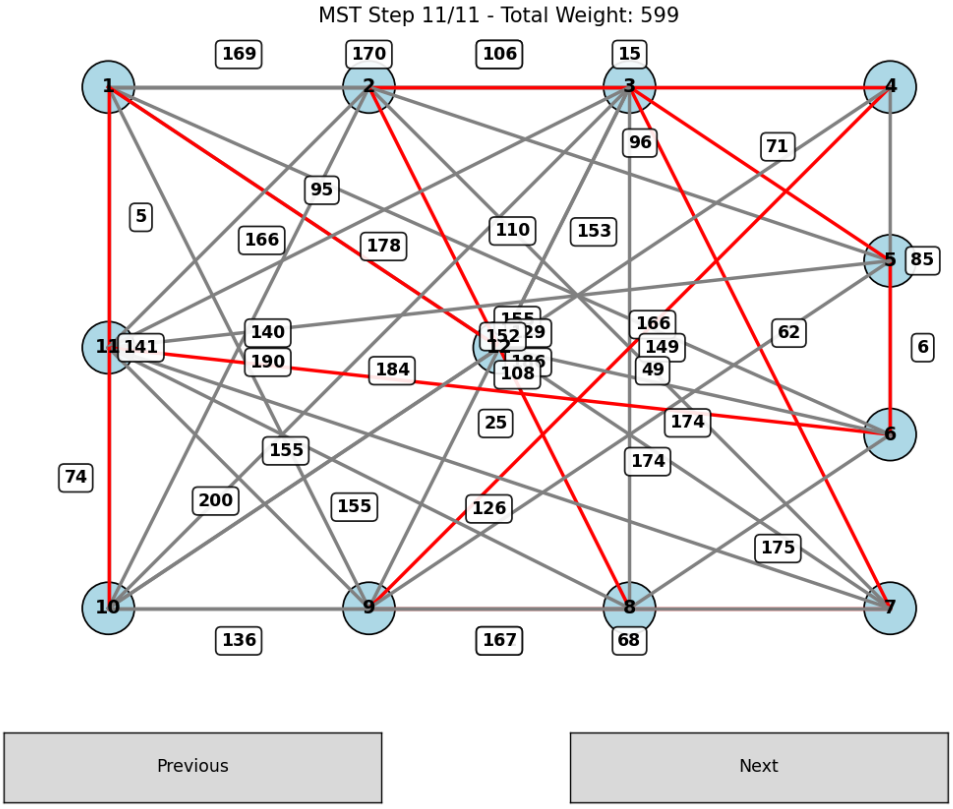
Next

MST Step 10/11 - Total Weight: 470



Previous

Next


MST Step 11/11 - Total Weight: 599
169
170
106
15
1
2
3
4
96
71
95
5
166
178
110
153
5
85
140
166
62
141
190
184
108
49
6
174
25
155
74
174
200
155
126
175
10
9
8
7
136
167
68
Previous
Next

Висновки:

У ході виконання лабораторної роботи було успішно розроблено програму для знаходження мінімального кістяка зваженого неорієнтованого графа за алгоритмом Пріма.

Практична реалізація алгоритму Пріма продемонструвала його ефективність для знаходження мінімального кістяка графа. Розроблений візуальний інтерфейс дозволяє наочно представити кроки алгоритму, що сприяє кращому розумінню його роботи. Програма коректно обробляє вхідні дані відповідно до варіанту завдання та правильно візуалізує результат.

Отриманий досвід дозволив закріпити теоретичні знання про алгоритми на графах та набути практичних навичок їх програмної реалізації з використанням сучасних інструментів візуалізації.