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-41

Димура Ілля Олександрович
Номер у списку групи: 7

Перевірив:

Сергієнко А. М.

Київ 2025

Завдання

1. Представити ненапрямлений граф із заданими параметрами так само, як у лабораторній роботі 3.

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

   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:

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

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

   3) одержується матриця D, у якій
      a) $d_{ij} = 0$, якщо $c_{ij} = 0$,
      b) $d_{ij} = 1$, якщо $c_{ij} > 0$, $d_{ij} \in D$, $c_{ij} \in C$;
   4) одержується матриця H, у якій
      a) $h_{ij} = 1$, якщо $d_{ij} != d_{ji}$,

         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 7:

$n_1n_2n_3n_4 = 4107$

Розміщення вершин: колом з вершиною в центрі

Кількість вершин: $10 + n_3 = 10$

Алгоритм знаходження мінімального кістяка: Пріма

Текст програми

```python
import math
import random
import tkinter as tk
from typing import List, Tuple

Coord = Tuple[float, float]
IntMatrix = List[List[int]]
FloatMatrix = List[List[float]]
Edge = Tuple[int, int, int]  # u, v, w

#### Configuration
VARIANT: int = 4107
n1, n2, n3, n4 = map(int, str(VARIANT).zfill(4))
####

VERTEX_CNT = 10 + n3
SEED = VARIANT
PANEL_SIZE = 600
PANEL_GAP = 40
OUTER_RADIUS = 0.40 * PANEL_SIZE
NODE_R = 22
EDGE_W = 3
```

```
def generate_directed_matrix(n: int) -> IntMatrix:
    random.seed(SEED)
    k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.05
    return [
        [1 if random.uniform(0.0, 2.0) * k >= 1.0 else 0 for _ in range(n)]
        for _ in range(n)
    ]


def to_undirected(mat: IntMatrix) -> IntMatrix:
    n = len(mat)
    res = [row[:] for row in mat]
    for i in range(n):
        for j in range(i + 1, n):
            res[i][j] = res[j][i] = 1 if mat[i][j] or mat[j][i] else 0
    return res


def generate_weight_matrix(adj: IntMatrix) -> IntMatrix:
    random.seed(SEED)
    n = len(adj)
    b: FloatMatrix = [[random.uniform(0.0, 2.0) for _ in range(n)] for _ in
range(n)]
    c: IntMatrix = [[math.ceil(b[i][j] * 100 * adj[i][j]) for j in range(n)] for i
in range(n)]
    d: IntMatrix = [[0 if c[i][j] == 0 else 1 for j in range(n)] for i in range(n)]
    h: IntMatrix = [[1 if d[i][j] != d[j][i] else 0 for j in range(n)] for i in
range(n)]
    tr: IntMatrix = [[1 if i < j else 0 for j in range(n)] for i in range(n)]
    weights: IntMatrix = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(i + 1, n):
            w = (d[i][j] + h[i][j] + tr[i][j]) * c[i][j]
```

```
            weights[i][j] = weights[j][i] = w
    return weights


def prim_mst(weights: IntMatrix) -> Tuple[List[Edge], int]:
    n = len(weights)
    in_tree = [False] * n
    dist = [math.inf] * n
    parent = [-1] * n

    dist[0] = 0
    total = 0
    edges: List[Edge] = []

    for _ in range(n):
        u = min((d, v) for v, d in enumerate(dist) if not in_tree[v])[1]
        in_tree[u] = True
        if parent[u] != -1:
            edges.append((parent[u], u, weights[u][parent[u]]))
            total += weights[u][parent[u]]

        for v in range(n):
            w = weights[u][v]
            if not in_tree[v] and 0 < w < dist[v]:
                dist[v] = w
                parent[v] = u
    return edges, total


def node_positions(n: int, center: int, ox: int) -> List[Coord]:
    cx, cy = ox + PANEL_SIZE / 2, PANEL_SIZE / 2
    pos = [None] * n  # type: ignore
```

```
    pos[center] = (cx, cy)
    outer = [i for i in range(n) if i ≠ center]
    for k, i in enumerate(outer):
        ang = 2 * math.pi * k / len(outer)
        pos[i] = (cx + OUTER_RADIUS * math.cos(ang),
                  cy + OUTER_RADIUS * math.sin(ang))
    return pos


def shift(p: Coord, q: Coord, d: float) → Coord:
    dx, dy = q[0] - p[0], q[1] - p[1]
    l = math.hypot(dx, dy)
    return (p[0] + dx / l * d, p[1] + dy / l * d)


def draw_node(cv: tk.Canvas, x: float, y: float, lbl: str):
    cv.create_oval(x - NODE_R, y - NODE_R, x + NODE_R, y + NODE_R,
                   fill="coral", width=2)
    cv.create_text(x, y, text=lbl, font=("Arial", 12, "bold"))


def draw_weight_label(cv: tk.Canvas, x: float, y: float, w: int):
    text_id = cv.create_text(x, y, text=str(w), fill="black", font=("Arial", 13))
    bbox = cv.bbox(text_id)
    if bbox:
        x1, y1, x2, y2 = bbox
        pad = 3
        rect_id = cv.create_rectangle(x1 - pad, y1 - pad, x2 + pad, y2 + pad,
                                      fill="white", outline="black")
        cv.tag_lower(rect_id, text_id)
```

```
def draw_edge(cv: tk.Canvas, p: Coord, q: Coord, w: int,
              color: str = "black", width: int = EDGE_W):
    a = shift(p, q, NODE_R)
    b = shift(q, p, NODE_R)

    options = {"fill": color, "width": width, "capstyle": tk.ROUND}
    line_id = cv.create_line(*a, *b, **options)

    cv.tag_lower(line_id)

    mid = (
        a[0] + 0.5 * (b[0] - a[0]),
        a[1] + 0.5 * (b[1] - a[1])
    )
    draw_weight_label(cv, mid[0], mid[1], w)


def print_matrix(mat: IntMatrix, title: str):
    n = len(mat)
    print(f"\n{title}")
    print("    " + " ".join(f"{j + 1:>3}" for j in range(n)))
    print("    " + "---" * n)
    for i in range(n):
        row = " ".join(f"{mat[i][j]:>3}" for j in range(n))
        print(f"{i + 1:>2}| {row}")


def print_edge_list(edges: List[Edge], title: str):
    print(f"\n{title}")
    for k, (u, v, w) in enumerate(edges, 1):
        print(f"Ребро {u}-{v}, вага: {w}")
```

```
def get_all_edges(weights: IntMatrix) -> List[Edge]:
    n = len(weights)
    return sorted([(i, j, weights[i][j]) for i in range(n) for j in range(i + 1, n)
if weights[i][j] > 0],
                  key=lambda e: e[2])



def main():
    directed = generate_directed_matrix(VERTEX_CNT)
    undirected = to_undirected(directed)
    weights = generate_weight_matrix(undirected)

    mst_edges, mst_weight = prim_mst(weights)
    all_edges = get_all_edges(weights)

    print_matrix(undirected, "Adjacency matrix (undirected)")
    print_matrix(weights, "Weight matrix W")
    print_edge_list(all_edges, "Усі ребра графа (за зростанням ваги)")
    print_edge_list(mst_edges, "Ребра мінімального кістяка (Прима)")

    print(f"\nАлгоритм: Прима")
    print("Ребра мін. кістяка:", [(u + 1, v + 1, w) for u, v, w in mst_edges])
    print("Сума ваг ребер МК =", mst_weight)

    cv_w = 2 * PANEL_SIZE + PANEL_GAP
    root = tk.Tk()
    root.title(f"Lab 6 · Variant {VARIANT}")
    cv = tk.Canvas(root, width=cv_w, height=PANEL_SIZE + 20, bg="white")
    cv.pack()

    pos_L = node_positions(VERTEX_CNT, VERTEX_CNT // 2, 0)
    pos_R = node_positions(VERTEX_CNT, VERTEX_CNT // 2, PANEL_SIZE + PANEL_GAP)
```

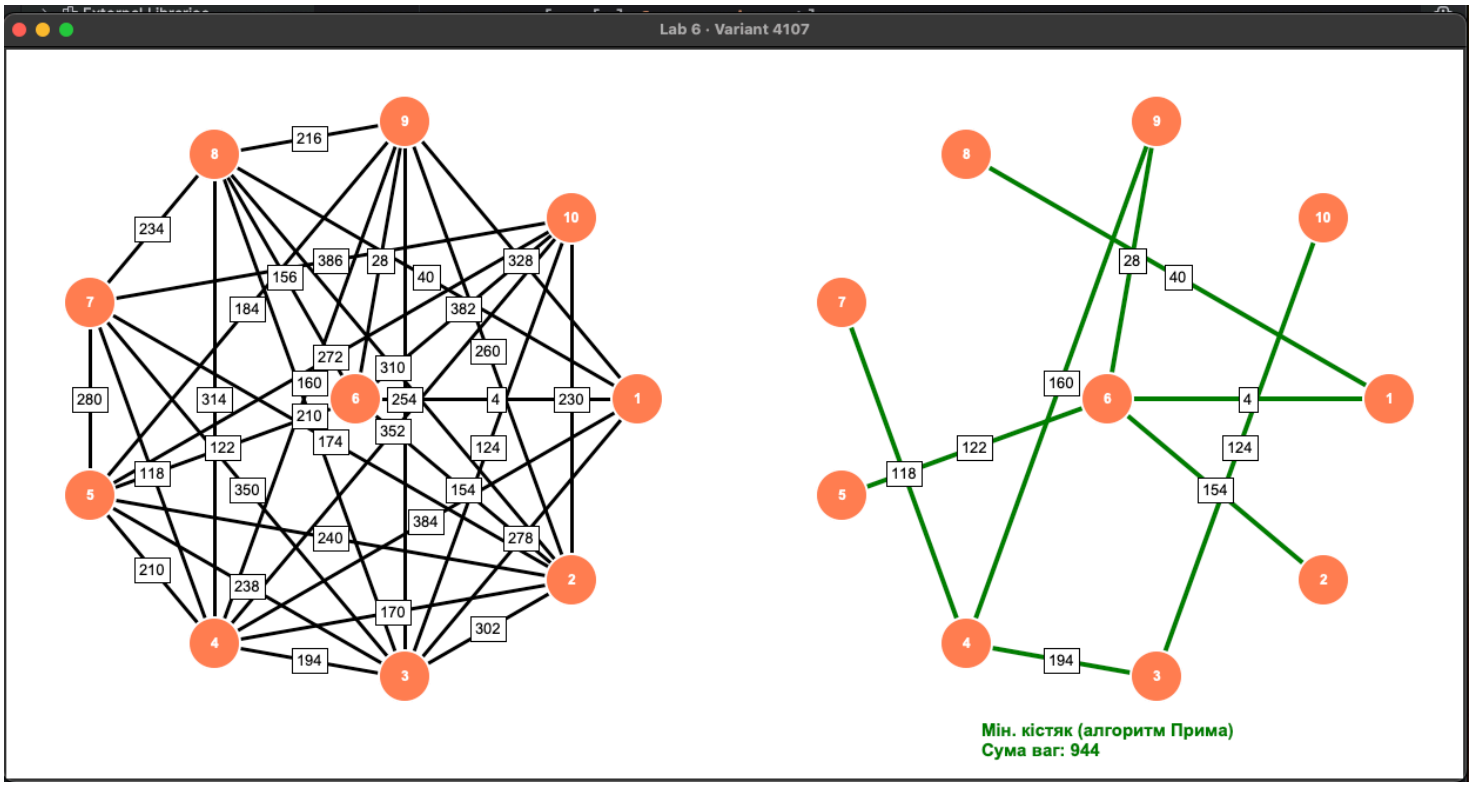
```
    # повний граф
    for i in range(VERTEX_CNT):
        for j in range(i + 1, VERTEX_CNT):
            if weights[i][j] > 0:
                draw_edge(cv, pos_L[i], pos_L[j], weights[i][j])
    for i, (x, y) in enumerate(pos_L):
        draw_node(cv, x, y, str(i + 1))
    cv.create_text(PANEL_SIZE / 2, PANEL_SIZE - 10,
                   text="Граф із вагами", font=("Arial", 15, "bold"))

    # мінімальний кістяк
    for u, v, w in mst_edges:
        draw_edge(cv, pos_R[u], pos_R[v], w, color="green", width=EDGE_W + 1)
    for i, (x, y) in enumerate(pos_R):
        draw_node(cv, x, y, str(i + 1))
    cv.create_text(PANEL_SIZE + PANEL_GAP + PANEL_SIZE / 2,
                   PANEL_SIZE - 10,
                   text=f"Мін. кістяк (алгоритм Прима)\nСума ваг: {mst_weight}",
                   font=("Arial", 15, "bold"), fill="green")

    root.mainloop()


if __name__ == "__main__":
    main()
```

Тести програми:




```
(asd-labs) ➜  6 git:(main) ✗ python3 main.py

Adjacency matrix (undirected)
    1   2   3   4   5   6   7   8   9  10
  ----------------------------
 1|   0   0   1   1   0   1   0   1   1   0
 2|   0   1   1   1   1   1   1   1   1   1
 3|   1   1   0   1   1   0   1   1   1   1
 4|   1   1   1   0   1   0   1   1   1   1
 5|   0   1   1   1   0   1   1   0   1   1
 6|   1   1   0   0   1   0   0   1   1   1
 7|   0   1   1   1   1   0   0   1   0   1
 8|   1   1   1   1   0   1   1   1   1   0
 9|   1   1   1   1   1   1   0   1   0   0
10|   0   1   1   1   1   1   1   0   0   0
```

```
Weight matrix W
     1   2   3   4   5   6   7   8   9  10
   ----------------------------
 1|   0   0 278 384   0   4   0  40 328   0
 2|   0   0 302 170 240 154 174 310 260 230
 3| 278 302   0 194 238   0 350 210 254 124
 4| 384 170 194   0 210   0 118 314 160 352
 5|   0 240 238 210   0 122 280   0 184 272
 6|   4 154   0   0 122   0   0 156  28 382
 7|   0 174 350 118 280   0   0 234   0 386
 8|  40 310 210 314   0 156 234   0 216   0
 9| 328 260 254 160 184  28   0 216   0   0
10|   0 230 124 352 272 382 386   0   0   0
```

```
Усі ребра графа (за зростанням ваги)
Ребро 0-5, вага: 4
Ребро 5-8, вага: 28
Ребро 0-7, вага: 40
Ребро 3-6, вага: 118
Ребро 4-5, вага: 122
Ребро 2-9, вага: 124
Ребро 1-5, вага: 154
Ребро 5-7, вага: 156
Ребро 3-8, вага: 160
Ребро 1-3, вага: 170
Ребро 1-6, вага: 174
Ребро 4-8, вага: 184
Ребро 2-3, вага: 194
Ребро 2-7, вага: 210
Ребро 3-4, вага: 210
Ребро 7-8, вага: 216
Ребро 1-9, вага: 230
Ребро 6-7, вага: 234
Ребро 2-4, вага: 238
Ребро 1-4, вага: 240
Ребро 2-8, вага: 254
Ребро 1-8, вага: 260
Ребро 4-9, вага: 272
Ребро 0-2, вага: 278
Ребро 4-6, вага: 280
Ребро 1-2, вага: 302
Ребро 1-7, вага: 310
Ребро 3-7, вага: 314
Ребро 0-8, вага: 328
Ребро 2-6, вага: 350
Ребро 3-9, вага: 352
Ребро 5-9, вага: 382
Ребро 0-3, вага: 384
Ребро 6-9, вага: 386
```

```
Ребра мінімального кістяка (Прима)
Ребро 0-5, вага: 4
Ребро 5-8, вага: 28
Ребро 0-7, вага: 40
Ребро 5-4, вага: 122
Ребро 5-1, вага: 154
Ребро 8-3, вага: 160
Ребро 3-6, вага: 118
Ребро 3-2, вага: 194
Ребро 2-9, вага: 124
```

Висновок:

Лабораторна робота була виконана на мові програмування Python, використовуючи кросплатформну бібліотеку tkinter. Був імплементований алгоритм Прима для побудови мінімального кістяка. Також додав логіку розрахунку ваг ребер та їх відображення. Алгоритм може використовуватись в багатьох задачах з реального світу (наприклад, проєктування мереж)