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 №2.4

з дисципліни

«Алгоритми і структури даних»

Виконав:
Студент групи ІМ-34

Никифоров Артем Михайлович

Номер у списку групи:16

Перевірила:
Молчанова А. А.

Київ 2024

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.

Завдання для конкретного варіанту:

n1n2n3n4 = 3416

Кількість вершин = 10 + n3 = 11

Розташування колом з вершиною в центрі, тому що n4 = 6

Текст програми:

```
import turtle
import math
import random
import copy

random.seed(3416)

matrix_dir = [[random.uniform(0.0, 2.0) for j in range(11)]
for i in range(11)]
matrix_B = [[random.uniform(0.0, 2.0) for j in range(11)]
for i in range(11)]

k = 1.0 - 1 * 0.001 - 6 * 0.005 - 0.05


for i in range(len(matrix_dir)):
    for j in range(len(matrix_dir[1])):
        matrix_dir[i][j] *= k
        if matrix_dir[i][j] < 1:
            matrix_dir[i][j] = 0
        else:
            matrix_dir[i][j] = 1

matrix2 = copy.deepcopy(matrix_dir)

for i in range(len(matrix2)):
    for j in range(len(matrix2[1])):
        if matrix_dir[i][j] == 1:
            matrix2[i][j] = 1
            matrix2[j][i] = 1
for i in range(len(matrix2)):
    print(matrix2[i], sep="\n")

matrix_C = [[math.ceil(matrix_B[i][j] * 100 * matrix2[i][j])
for j in range(len(matrix_B[0]))] for i in
range(len(matrix_B))]

matrix_D = [[1 if matrix_C[i][j] > 0 else 0 for j in
range(len(matrix_C[0]))] for i in range(len(matrix_C))]

matrix_H = [[1 if matrix_D[i][j] != matrix_D[j][i] else 0
for j in range(len(matrix_D[0]))] for i in
range(len(matrix_D))]
```

```
Tr = [[1,1,1,1,1,1,1,1,1,1,1],
      [0,1,1,1,1,1,1,1,1,1,1],
      [0,0,1,1,1,1,1,1,1,1,1],
      [0,0,0,1,1,1,1,1,1,1,1],
      [0,0,0,0,1,1,1,1,1,1,1],
      [0,0,0,0,0,1,1,1,1,1,1],
      [0,0,0,0,0,0,1,1,1,1,1],
      [0,0,0,0,0,0,0,1,1,1,1],
      [0,0,0,0,0,0,0,0,1,1,1],
      [0,0,0,0,0,0,0,0,0,1,1],
      [0,0,0,0,0,0,0,0,0,0,1]]

W = [[0 for _ in range(len(matrix_C))] for _ in
range(len(matrix_C))]

for i in range(len(matrix_C)):
    for j in range(len(matrix_C)):
        W[i][j] = (matrix_D[i][j] + matrix_H[i][j] *
Tr[i][j]) * matrix_C[i][j]
        W[j][i] = W[i][j]

print("Матриця W")
for row in W:
    print(row)

screen = turtle.Screen()
screen.setup(width=600, height=600)
screen.bgcolor("white")
turtle.speed(0)
turtle.hideturtle()

dir_check = int(input("choose graph to output 0 = undir, 1 =
dir \n"))

def draw_circles_in_circle():
    radius = 200
    num_circles = 10
    angle = 360 / num_circles
    for i in range(num_circles):
        x = radius * math.cos(math.radians(angle * i))
        y = radius * math.sin(math.radians(angle * i))
        turtle.penup()
        turtle.color('black')
```

```
        turtle.goto(x, y)
        turtle.pendown()
        turtle.begin_fill()
        turtle.circle(20)
        turtle.end_fill()
        turtle.penup()
        turtle.goto(x, y + 10)
        turtle.color('white')
        turtle.write(str(i+1), align="center",
font=("Arial", 12, "normal"))
        turtle.penup()

def draw_11():
    turtle.color('black')
    turtle.penup()
    turtle.goto(0, 0)
    turtle.pendown()
    turtle.begin_fill()
    turtle.circle(20)
    turtle.end_fill()
    turtle.color('white')
    turtle.penup()
    turtle.goto(0, 0 + 10)
    turtle.write(str(11), align="center", font=("Arial", 12,
"normal"))
    turtle.penup()

def draw_edges_undir(matrix2): #lines of undir matrix
    num_vertices = len(matrix2)
    for i in range(num_vertices):
        for j in range(i, num_vertices):
            if matrix2[i][j] == 1:
                x1, y1 = get_vertex_position(i)
                x2, y2 = get_vertex_position(j)
                if i == j:
                    draw_circle(x1, y1)
                else:
                    if (i == 2 and j == 7) or (i == 7 and j
== 2) or (i == 1 and j == 6) or (i == 3 and j == 8) or (i ==
4 and j == 9):
                        cursed_line(x1, y1, x2, y2, W[i][j])
                    else:
                        draw_line(x1, y1, x2, y2, W[i][j])
```

```
def draw_edges_dir(matrix_dir):  # lines of dir matrix
    num_vertices = len(matrix_dir)
    for i in range(num_vertices):
        for j in range(num_vertices):
            if matrix_dir[i][j] == 1:
                x1, y1 = get_vertex_position(i)
                x2, y2 = get_vertex_position(j)
                if i == j:
                    draw_circle_dir(x1, y1)
                else:
                    if matrix_dir[j][i] == 1:
                        cursed_line_dir(x1, y1, x2, y2,
W[i][j])
                    elif (i == 7 and j == 2) or (i == 1 and
j == 6) or (i == 8 and j == 3) or (i == 9 and j == 4):
                        cursed_line_dir(x1, y1, x2, y2,
W[i][j])
                    else:
                        draw_dir_line(x1, y1, x2, y2,
W[i][j])

def cursed_line(x1, y1, x2, y2, weight):
    turtle.penup()
    turtle.goto(x1, y1 + 15)
    turtle.pendown()
    turtle.color('red')
    turtle.width(1)

    if x1 == x2:
        control_offset = 15
        cx1, cy1 = (x1 + x2) / 2, (y1 + y2) / 2
        if y1 < y2:
            cy1 += control_offset
        else:
            cy1 -= control_offset
        turtle.goto(cx1, cy1 + 15)
    else:
        control_offset = 17
        cx1, cy1 = (x1 + x2) / 2, (1.2*y1 + y2) / 2
        if x1 < x2:
            cx1 += control_offset
        else:
            cx1 -= control_offset
        turtle.goto(cx1, cy1 + 15)
```

```
    turtle.goto(x2, y2 + 15)
    draw_weight((x1 + x2) / 2, (y1 + y2) / 2, weight)

def cursed_line_dir(x1, y1, x2, y2):
    turtle.penup()
    turtle.goto(x1, y1)
    turtle.pendown()
    turtle.color('red')
    turtle.width(1)

    if x1 == x2:
        control_offset = 15
        cx1, cy1 = (x1 + x2) / 2, (y1 + y2) / 2
        if y1 < y2:
            cy1 += control_offset
        else:
            cy1 -= control_offset
        turtle.goto(cx1, cy1)
    else:
        control_offset = 15
        cx1, cy1 = (x1 + x2) / 2, (1.2*y1 + y2) / 2
        if x1 < x2:
            cx1 += control_offset
        else:
            cx1 -= control_offset
        turtle.goto(cx1, cy1)
    turtle.goto(x2, y2)
    turtle_angle = math.degrees(math.atan2(y2 - y1, x2 -
x1))
    turtle.setheading(turtle_angle)
    turtle.stamp()

class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

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

    def union(self, u, v):
        root_u = self.find(u)
```

```
        root_v = self.find(v)

        if root_u != root_v:
            if self.rank[root_u] > self.rank[root_v]:
                self.parent[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.parent[root_u] = root_v
            else:
                self.parent[root_v] = root_u
                self.rank[root_u] += 1

def kruskal_mst(graph):
    edges = []
    for i in range(len(graph)):
        for j in range(i + 1, len(graph[i])):
            if graph[i][j] != 0:
                edges.append((graph[i][j], i, j))

    edges.sort()
    ds = DisjointSet(len(graph))
    mst = []
    for edge in edges:
        weight, u, v = edge
        if ds.find(u) != ds.find(v):
            ds.union(u, v)
            mst.append(edge)
    return mst

def draw_circle(x, y):
    turtle.color('black')
    turtle.penup()
    turtle.goto(x, y+20)
    turtle.pendown()
    turtle.circle(30)
    turtle.penup()

def draw_circle_dir(x, y):
    turtle.color('blue')
    turtle.penup()
    turtle.goto(x, y+20)
    turtle.pendown()
    turtle.circle(30)
    turtle.stamp()
    turtle.penup()
```

```
def get_vertex_position(vertex_index):
    if vertex_index == 10:
        return 0, 0
    else:
        radius = 200
        num_vertices = 10
        angle = 360 / num_vertices
        x = radius * math.cos(math.radians(angle *
vertex_index))
        y = radius * math.sin(math.radians(angle *
vertex_index))
        return x, y

def draw_line(x1, y1, x2, y2, weight):
    turtle.penup()
    turtle.goto(x1, y1+15)
    turtle.pendown()
    turtle.color('black')
    turtle.width(1)
    turtle.goto(x2, y2+15)
    draw_weight((x1 + x2) / 2, (y1 + y2) / 2, weight)

def draw_dir_line(x1, y1, x2, y2, weight):
    turtle.penup()
    turtle.goto(x1, y1+15)
    turtle.pendown()
    turtle.color('blue')
    turtle.width(1)
    end_x_line = x1 + 0.95 * (x2 - x1)
    end_y_line = y1 + 0.95 * (y2 - y1)
    turtle.goto(end_x_line, end_y_line + 15)
    turtle_angle = math.degrees(math.atan2(end_y_line - y1,
end_x_line - x1))
    turtle.setheading(turtle_angle)
    turtle.stamp()
    draw_weight((x1 + x2) / 2, (y1 + y2) / 2, weight)

def draw_weight(x, y, weight):
    turtle.penup()
    turtle.goto(x, y)
    turtle.pendown()
    turtle.color('red')
```

```
    turtle.write(f"{weight:.1f}", align="center",
font=("Arial", 10, "normal"))
    turtle.penup()

def draw_traversed_edge(x1, y1, x2, y2, curve=False):
    turtle.penup()
    turtle.goto(x1, y1 + 15)
    turtle.pendown()
    turtle.color('green')
    turtle.width(2)

    if curve:
        control_offset = 15
        cx1, cy1 = (x1 + x2) / 2, (y1 + y2) / 2
        if y1 < y2:
            cy1 += control_offset
        else:
            cy1 -= control_offset
        turtle.goto(cx1, cy1 + 15)

    turtle.goto(x2, y2 + 15)

def visualize_kruskal_mst(mst):
    for weight, u, v in mst:
        x1, y1 = get_vertex_position(u)
        x2, y2 = get_vertex_position(v)

        turtle.penup()
        turtle.goto(x1, y1)
        turtle.pendown()
        turtle.color('green')
        turtle.begin_fill()
        turtle.circle(5)
        turtle.end_fill()

        draw_traversed_edge(x1, y1, x2, y2)

        turtle.penup()
        turtle.goto(x2, y2)
        turtle.pendown()
        turtle.color('green')
        turtle.begin_fill()
        turtle.circle(5)
        turtle.end_fill()
```

```
        input("Натискайте Enter щоб зробити наступний крок")

mst = kruskal_mst(W)
print("Ваги ребер використовуючи Алгоритм Краскала:")
for weight, u, v in mst:
    print(f"Ребро між {u+1} і {v+1} з вагою {weight}")

if dir_check == 1:
    draw_circles_in_circle()
    draw_11()
    draw_edges_dir(matrix_dir)
    num_vertices = 10
else:
    draw_edges_undir(matrix2)
    draw_circles_in_circle()
    num_vertices = 10
    draw_11()

visualize_kruskal_mst(mst)

screen.exitonclick()
```

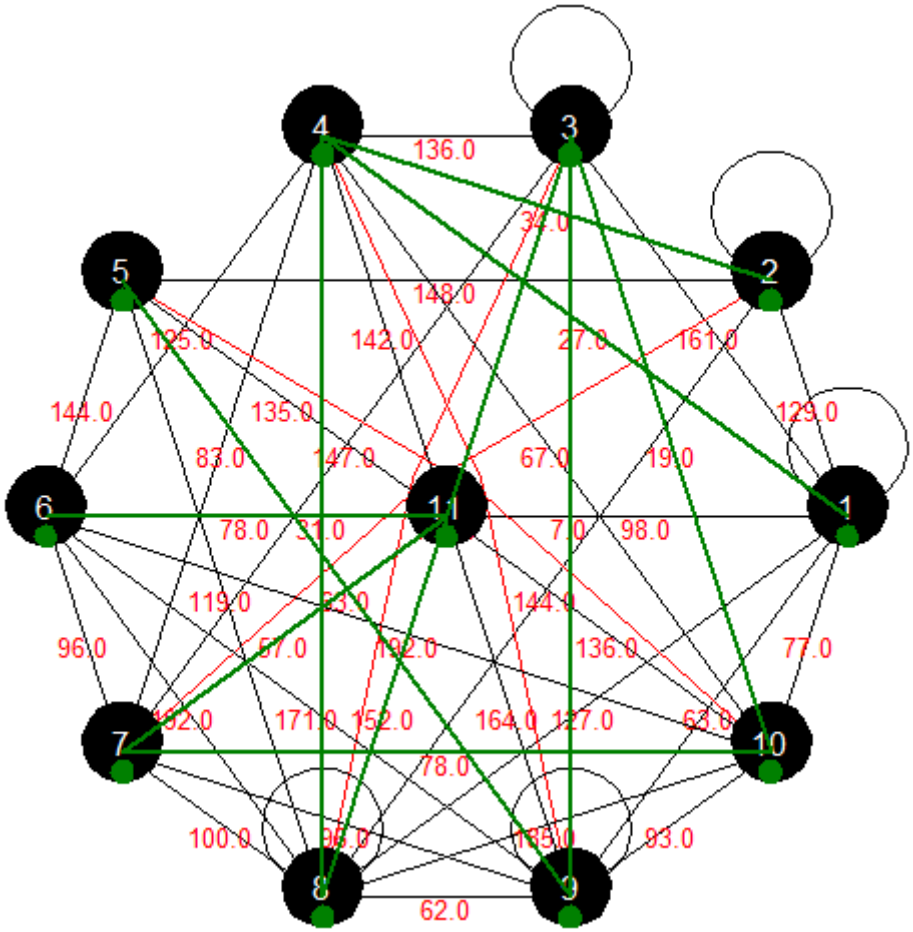
Згенерована матриця суміжності ненапрямленого графа:

```
[1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0]
[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1]
[0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1]
[1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
```

Згенерована матриця ваг графа:

```
Матриця W
[117, 129, 161, 27, 0, 0, 0, 127, 63, 77, 98]
[129, 103, 0, 34, 148, 0, 191, 144, 0, 0, 0]
[161, 0, 94, 136, 0, 0, 147, 9, 7, 19, 0]
[27, 34, 136, 0, 0, 125, 83, 31, 144, 67, 142]
[0, 148, 0, 0, 0, 144, 0, 119, 63, 141, 135]
[0, 0, 0, 125, 144, 0, 96, 192, 171, 192, 78]
[0, 191, 147, 83, 0, 96, 0, 100, 96, 78, 57]
[127, 144, 9, 31, 119, 192, 100, 184, 62, 185, 152]
[63, 0, 7, 144, 63, 171, 96, 62, 14, 93, 164]
[77, 0, 19, 67, 141, 192, 78, 185, 93, 0, 136]
[98, 0, 0, 142, 135, 78, 57, 152, 164, 136, 0]
```

Скриншоти зображення графа та його мінімального кістяка:



Сума ваг ребер знайденого мінімального кістяка:

```
Ваги ребер використовуючи Алгоритм Краскала:
Ребро між 3 і 9 з вагою 7
Ребро між 3 і 8 з вагою 9
Ребро між 3 і 10 з вагою 19
Ребро між 1 і 4 з вагою 27
Ребро між 4 і 8 з вагою 31
Ребро між 2 і 4 з вагою 34
Ребро між 7 і 11 з вагою 57
Ребро між 5 і 9 з вагою 63
Ребро між 6 і 11 з вагою 78
Ребро між 7 і 10 з вагою 78
```

Висновок:
Алгоритм Краскала доволі простий у реалізації. Алгоритм Краскала особливо ефективний для графів із великою кількістю ребер, оскільки він спочатку сортує всі ребра, а потім обробляє їх у порядку зростання ваги.