**Міністерство освіти і науки України**
**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**
**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації і управління**

# Звіт

з лабораторної роботи № 3 з дисципліни
«Алгоритми та структури даних 3. Структури даних»

**«Прикладні задачі теорії графів ч.1»**

**Варіант 20**

**“Побудова мінімальних покриваючих дерев методом Крускала”**

**Виконала:** **ІП-13 Лисенко Анастасія Олегівна**

**Перевірив:** **Сопов Олексій Олександрович**

Київ 2022

ЗМІСТ

# 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні прикладні алгоритми на графах та способи їх імплементації.

# 2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором). Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв'язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв'язання задачі.

Таблиця 2.1 – Варіанти алгоритмів

| **№** | **Задача** | **Алгоритм** | **Тип графу** | **Спосіб задання графу** |
|---|---|---|---|---|
| 1 | Обхід графу | DFS | Неорієнтований | Матриця суміжності |
| 2 | Обхід графу | BFS | Неорієнтований | Матриця суміжності |
| 3 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця суміжності |
| 4 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця суміжності |
| 5 | Пошук найкоротшого шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |

| | | | | |
|---|---|---|---|---|
| 6 | Пошук найкоротшого шляху між парою вершин | Беллмана Форда | Орієнтований | Матриця вагів |



| | | | | |
|---|---|---|---|---|
| 7 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 8 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 9 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 10 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця суміжності |
| 11 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця суміжності |
| 12 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця суміжності |
| 13 | Обхід графу | DFS | Неорієнтований | Матриця інцидентності |
| 14 | Обхід графу | BFS | Неорієнтований | Матриця інцидентності |

| | | | | |
|---|---|---|---|---|
| 15 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця інцидентності |
| 16 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця інцидентності |
| 17 | Пошук найкоротшого шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |
| 18 | Пошук | Беллмана- | Орієнтований | Матриця вагів |



| | | | | |
|---|---|---|---|---|
| | найкоротшого шляху між парою вершин | Форда | | |
| 19 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 20 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 21 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 22 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця інцидентності |

| | | | | |
|---|---|---|---|---|
| 23 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця інцидентності |
| 24 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця інцидентності |
| 25 | Обхід графу | DFS | Неорієнтований | Матриця суміжності |
| 26 | Обхід графу | BFS | Неорієнтований | Матриця суміжності |
| 27 | Пошук маршруту у графі | Террі | Неорієнтований | Матриця суміжності |
| 28 | Пошук відстані між вершинами графа | Хвильовий | Неорієнтований | Матриця суміжності |
| 29 | Пошук найкоротшого | Дейкстри | Орієнтований | Матриця вагів |



| | | | | |
|---|---|---|---|---|
| | шляху між парою вершин | | | |
| 30 | Пошук найкоротшого шляху між парою вершин | Беллмана Форда | Орієнтований | Матриця вагів |

| | | | | |
|---|---|---|---|---|
| 31 | Побудова мінімальних покриваючих дерев | Прима | Неорієнтований | Матриця вагів |
| 32 | Побудова мінімальних покриваючих дерев | Крускала | Неорієнтований | Матриця вагів |
| 33 | Побудова мінімальних покриваючих дерев | Борувки | Неорієнтований | Матриця вагів |
| 34 | Побудова Ейлерового циклу | За означенням | Неорієнтований | Матриця суміжності |
| 35 | Побудова Ейлерового циклу | Флері | Неорієнтований | Матриця суміжності |
| 36 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця суміжності |

# 3 ВИКОНАННЯ

## 3.1 Псевдокод алгоритму

```
result = []
i, e = 0, 0
self.graph = sorted(self.graph, key=lambda item: item[2]) #sorting
parent = []
rank = []
for node in range(self.V) do
  parent.append(node)
  rank.append(0)
end for
while e < self.V - 1 do
  start, end, weight = self.graph[i]
  i = i + 1
  x = self.find(parent, start)
  y = self.find(parent, end)
  if x != y then
    e = e + 1
    result.append([start, end, weight])
    self.apply_union(parent, rank, x, y)
   end if
end while
return result
```

## 3.2 Програмна реалізація алгоритму

### 3.2.1 Вихідний код

```
from networkx.generators.random_graphs import
fast_gnp_random_graph
import random
import networkx as nx
import matplotlib.pyplot as plt
import pylab


class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def add_edge(self, start, end, weight):
        self.graph.append([start, end, weight])
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def apply_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_algo(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:
            start, end, weight = self.graph[i]
            i = i + 1
            x = self.find(parent, start)
            y = self.find(parent, end)
            if x != y:
                e = e + 1
                result.append([start, end, weight])
                self.apply_union(parent, rank, x, y)
        return result
```

```
def create_weighted_graph(n):
    p = 0.5
    w_edges = []
    l = [0, 0, 0]
    mode = int(input("Enter the mode: 1 - if you want random
weights; 2 - if you want to input it yourself: "))
    gr = fast_gnp_random_graph(n, p)
    print("The list of all edges:", gr.edges)
    uw_edges = list(gr.edges())
    length = len(gr.edges)
    for i in range(0, length):
        l = [0, 0, 0]
        for j in range(0, 3):
            if j == 2:
                if mode == 1:
                    l[j] = random.randint(1, 7)
                if mode == 2:
                    l[j] = int(input("Enter the weight of the
edge: "))
            else:
                l[j] = uw_edges[i][j]
        w_edges.append(l)
    return w_edges, length

def print_graph(w_edges, length, res):

    G = nx.Graph()
    for i in range(0, length):
G.add_edges_from([(str(w_edges[i][0]), str(w_edges[i][1]))],
weight = w_edges[i][2])
    edge_labels=dict([((u,v,),d['weight'])
                      for u,v,d in G.edges(data=True)])
    red_edges = []
    for i in range(0, len(res)):
        red_edges.append((str(res[i][0]), str(res[i][1])))
    edge_colors = ['r' if edge in red_edges else 'b' for edge
in G.edges()]
    pos=nx.circular_layout(G)
    node_labels = {node:node for node in G.nodes()}
    nx.draw_networkx_labels(G, pos, labels=node_labels)
    nx.draw_networkx_edge_labels(G, pos,
edge_labels=edge_labels)
    nx.draw(G, pos, node_color='blue', node_size=1200,
edge_color=edge_colors, edge_cmap=plt.cm.Reds)
    pylab.show()
```

```
def main():
    n = int(input("Enter the amount of nodes:"))
    g = Graph(n)
    w_edges, length = create_weighted_graph(n)
    for i in range(0, length):
        g.add_edge(w_edges[i][0], w_edges[i][1],
w_edges[i][2])
    print("The minimal tree:\n")
    res = g.kruskal_algo()
    res_weight = 0
    res_len = len(res)
    for start, end, weight in res:
        res_weight += weight
        print("%d - %d: %d" % (start, end, weight))
    print('The weight of the tree:', res_weight)
    print_graph(w_edges, length, res)

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

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

```
Enter the amount of nodes:7
Enter the mode: 1 - if you want random weights; 2 - if you want to input it yourself: 1
The list of all edges: [(0, 1), (0, 2), (0, 3), (1, 2), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 5), (4, 5), (4, 6), (5, 6)]
The minimal tree:

0 - 3: 1
3 - 5: 1
4 - 6: 1
2 - 3: 2
2 - 4: 2
0 - 1: 4
The weight of the tree: 11
```

Також додаю фото побудованого графу для наочності, але чомусь не всі сторони, які входять у мінімальне покриваюче дерево фарбуються червоним, але сама реалізація алгоритму працює коректно

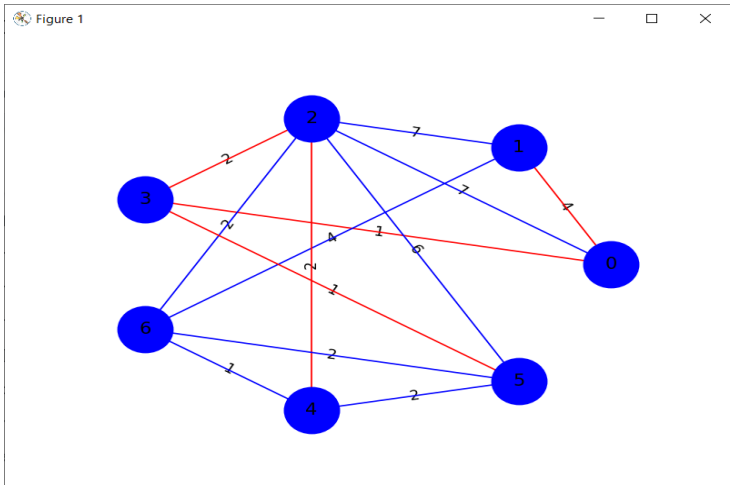
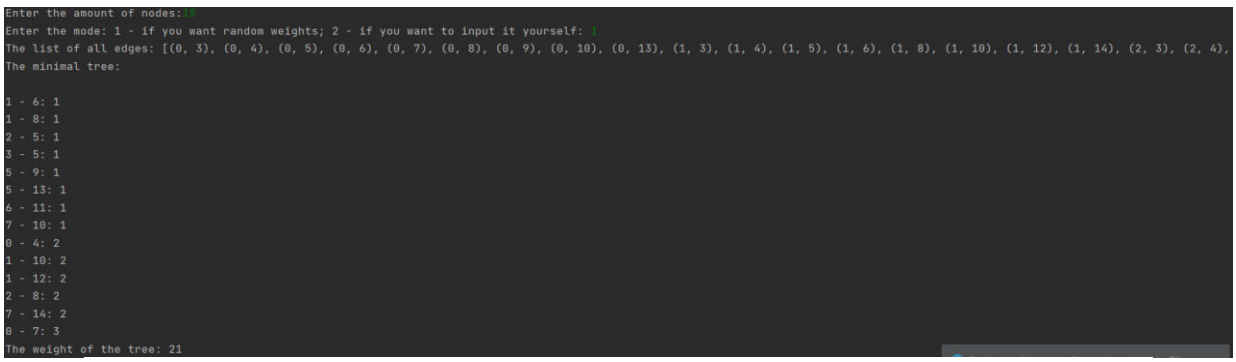
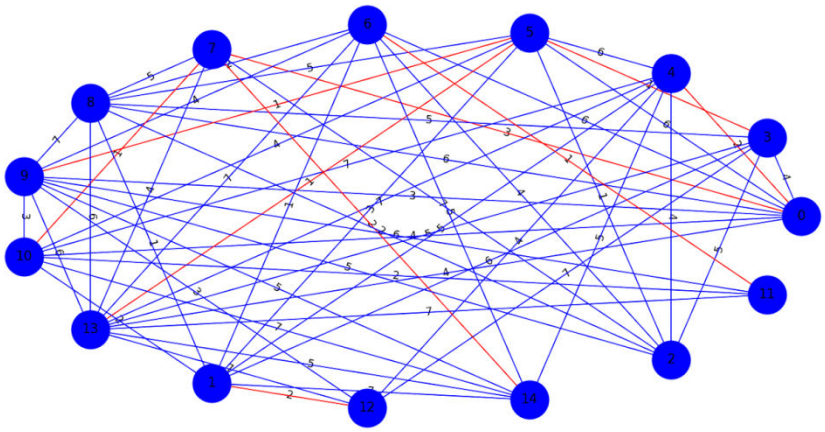



Рисунок 3.1 – Задача 1



Знову ж таки, чомусь не всі сторони промалювалися, але алгоритм пошуку працює корректно

Рисунок 3.2 – Задача 2

3.3 Розв’язання задачі вручну

На рисунку 3.3 наведено розв’язання задачі вручну.

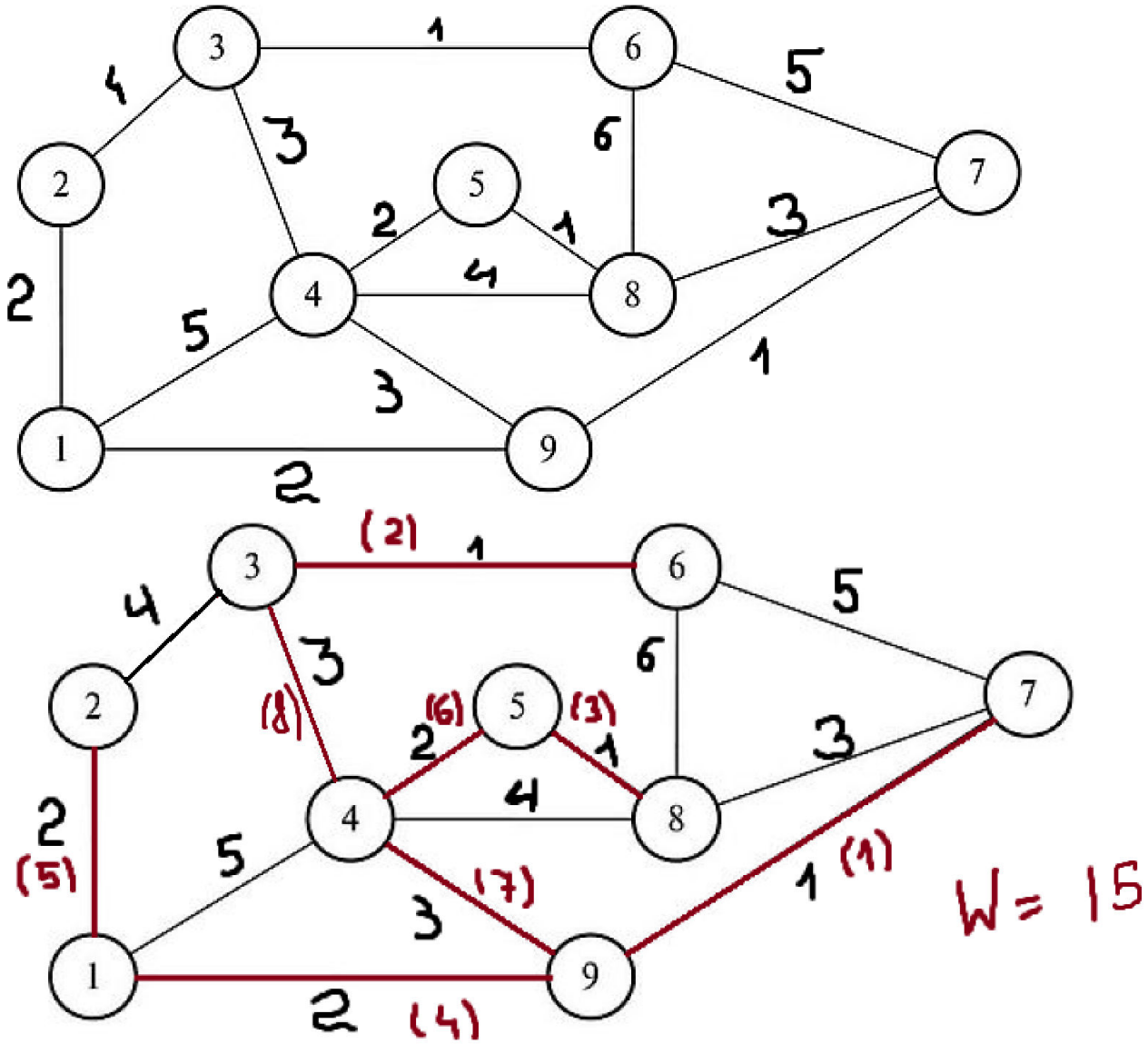



Рисунок 3.3 – Розв’язання задачі вручну

# ВИСНОВОК

При виконанні даної лабораторної роботи я вивчила основні прикладні алгоритми на графах та способи їх імплементації, написала на протестувала роботу алгоритму Крускела для побудови мінімальних покриваючих дерев, навчилася вирішувати задачі вручну за допомогою цього алгоритму.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 15.03.2022 включно максимальний бал дорівнює – 5. Після 15.03.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

– псевдокод алгоритму – 10%;

– програмна реалізація алгоритму – 50%;

– розв’язання задачі вручну – 20%;

– відповідь на 3 теоретичні питання по темі роботи 15%

– висновок – 5%.

НЕ ДІЮТЬ