Тема: Графи сум і різницеві графи (sum graphs and difference graphs)

Науковий керівник: к. ф.-м. н. Козеренко С.О.

Виконав: студент 4-го курсу спеціальності “Прикладна математика” Севергін Олександр Вадимович

Постановка задачі:

1. Дослідити властивості графів сум і різницевих графів.
2. Довести основні твердження.
3. Розробити алгоритми маркування графів сум мовою програмування Python.
4. Розробити загальне маркування для оливкових дерев.
5. Розробити загальне маркування для дерев-павуків.

Основні означення:

Маркуванням на графі G(V,E) називається функція вигляду σ: V -> ℕ ∪ {0}.

Граф суми G(V,E) - це граф G, у якому кожному ребру {u,v} Є E(G) відповідає вершина w така, що σ(u) + σ(v) = σ(w).

Різницевий граф G(V,E) - це граф G, у якому кожному ребру {u,v} Є E(G) відповідає вершина w така, що |σ(u) - σ(v)| = σ(w).

Алгоритми розмітки дерев:

Algorithm 3 Наївний алгоритм для розмітки графів-гусеней.

[5] Довільно позначте деяку вершину $v_0 \in V(T)$ додатним цілим числом α. Виберемо v_1 сусіда з v_0 та позначимо v_1 β, де $\beta > \alpha$ і $\beta \neq 2\alpha$ (так що $\sigma(v_0)+\sigma(v_0) \neq \sigma(v_1)$). Тепер, оскільки v_0v_1 є ребром, $\sigma(?_0) + \sigma(v_1)$ має з'явитися як мітка деякої вершини. Виберемо суміжну непозначену вершину v_2 до вже позначеної вершини u_2 ($u_2 = v_0$ або v_1), і позначимо v_2 за допомогою $\sigma(v_0) + \sigma(v_1)$. Тепер $\sigma(v_2) + \sigma(u_2)$ має з'явитися на деякій вершині, тому оберемо непозначену вершину v_3, що суміжна з вже позначеною вершиною u_3, і позначає її $?(v_2) + \sigma(u_2)$. Продовжуємо таким чином: на кожному кроці вершина v_{i+1} отримує мітку $\sigma(v_i) + \sigma(u_i)$, де u_i є єдиним позначеним сусідом v. Коли позначається остання вершина $v_{n-1} \in T$, сума $\sigma(v_{n-1}) + \sigma(U_{n-1})$ розподіляється ізольованій вершині z.

Алгоритми розмітки дерев:

Algorithm 4 Алгоритм графа гусені.

[5] Припустимо, що кожна вершина хребта s_i графа-гусені C має стопи $t_{ij}, 1 \leq j \leq fi$, що прилягають до неї. Маємо $f_i \geq 0$ для $1 \leq i \leq l-1$, а $f_0 = f_i = 0$ - хвіст і голова гусениці є листям. Використовуємо **Наївний Алгоритм**, щоб позначити вершини в порядку: $s_0, s_1, t_{11}, t_{12}, \ldots, t_{1f_1}, s_2, t_{21}, t_{22}, t_{2f_2}, s_3, \ldots, s_l$ (за ним z).

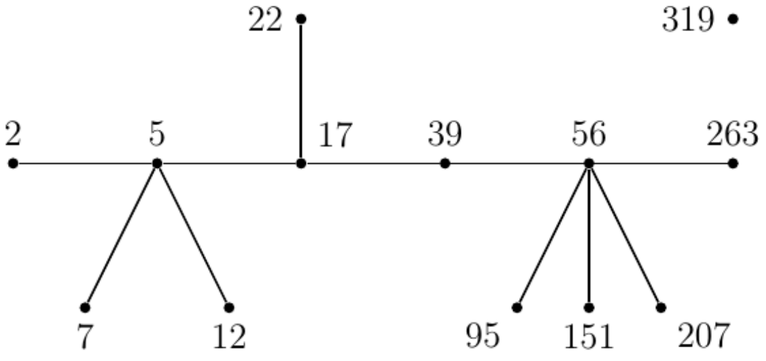


Рис. 2.5.2. Граф гусінь T ромаркований за **Алгоритмом 4.**

Реалізація алгоритму розмітки суми для графів-гусеней

1. Перевірка чи задана послідовність є кодом Прюфера.
2. Перевірка чи задана послідовність є графом-гусенем.
3. Пронумерувати вершини, використовуючи маркування суми для графів-гусеней.

Алгоритм розмітки суми для графів-гусеней

```python
import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict, deque
```

```python
def is_prufer_sequence(seq):
    """Tests whether the given sequence is a valid Prüfer sequence.

    A Prüfer sequence for a tree with n vertices is a sequence of n-2 numbers,
    which determines the structure of the tree.

    Parameters:
    seq (list): A list of integers representing a Prüfer sequence.

    Returns:
    bool: True if seq is a valid Prüfer sequence, False otherwise.
    """
    if not isinstance(seq, list):
        raise ValueError("Вхідні дані мають бути списком цілих чисел.")

    if not all(isinstance(x, int) for x in seq):
        raise ValueError("Усі елементи списку мають бути цілими числами.")

    if len(seq) == 0:
        raise ValueError("Послідовність не може бути порожньою.")

    n = len(seq) + 2

    for node in seq:
        if node < 1 or node >= n:
            raise ValueError('Усі вершини в послідовності Прюфера мають бути більші за 0 і менші за n.')
    degree = [1] * n
```

```python
for node in seq:
    degree[node - 1] += 1

for node in seq:
    found = False
    for i in range(1, n + 1):
        if degree[i - 1] == 1:
            degree[i - 1] -= 1
            degree[node - 1] -= 1
            found = True
            break
    if not found:
        return False

return True
```

Алгоритм розмітки суми для графів-гусеней

```python
def prufer_sequence_to_tree(seq):
    """Converts a Prüfer sequence into a tree represented as a NetworkX graph.

    Parameters:
    prufer_sequence (list): A list of integers representing the Prüfer sequence.

    Returns:
    G (networkx.Graph): A NetworkX graph representing the tree.
    """
    if not isinstance(seq, list):
        raise ValueError("Вхідні дані мають бути списком цілих чисел.")

    if not all(isinstance(x, int) for x in seq):
        raise ValueError("Усі елементи списку мають бути цілими числами.")

    if len(seq) == 0:
        raise ValueError("Послідовність не може бути порожньою.")

    if len(seq) >= 2:
        n = len(seq) + 2
    else:
        raise ValueError("Недостатня кількість елементів у послідовності для створення де

    for node in seq:
        if node < 1 or node > n:
            raise ValueError('Усі вершини в послідовності Прюфера мають бути більші за 0
```

```python
if len(seq) >= 2:
    n = len(seq) + 2
else:
    raise ValueError("Недостатня кількість елементів у послідовності для створення дерева.")

for node in seq:
    if node < 1 or node > n:
        raise ValueError('Усі вершини в послідовності Прюфера мають бути більші за 0 і менші за n.')

degree = [1] * n
for node in seq:
    degree[node - 1] += 1
edges = []
for node in seq:
    for i in range(n):
        if degree[i] == 1:
            edges.append((i + 1, node))
            degree[i] -= 1
            degree[node - 1] -= 1
            break

u, v = [i + 1 for i in range(n) if degree[i] == 1]
edges.append((u, v))

return edges
```

Алгоритм розмітки суми для графів-гусеней

```python
def is_caterpillar(edges):
    """Tests whether the given list of edges represents a caterpillar graph.

    A caterpillar is a tree in which the removal of all leaves leaves a path.

    Parameters:
    edges (list of tuples): List of edges in the form of tuples (u, v), where u and v are vertices.

    Returns:
    bool: True if the graph is a worm graph, False otherwise.
    """
    if not edges:
        return False

    adj_list = defaultdict(list)
    degrees = defaultdict(int)

    for u, v in edges:
        adj_list[u].append(v)
        adj_list[v].append(u)
        degrees[u] += 1
        degrees[v] += 1

    leaves = [node for node in degrees if degrees[node] == 1]

    if not leaves:
      return False
```

```python
if not leaves:
  return False

while leaves:
    new_leaves = []
    for leaf in leaves:
        for neighbor in adj_list[leaf]:
            if neighbor in degrees:
                degrees[neighbor] -= 1
            if neighbor in degrees and degrees[neighbor] == 1:
                new_leaves.append(neighbor)
        if degrees[leaf] != 0:
            degrees.pop(leaf)
    leaves = new_leaves

remaining_nodes = [node for node in degrees]

if not remaining_nodes:
    return False
if len(remaining_nodes) == 1:
    return True
if len(remaining_nodes) == 2:
    return degrees[remaining_nodes[0]] == 1 and degrees[remaining_nodes[1]] == 1

return all(degrees[node] == 2 for node in remaining_nodes)
```

Алгоритм розмітки суми для графів-гусеней

```python
def find_caterpillar_backbone(adj_list):
    """
    This function finds the backbone nodes of a caterpillar graph based on its adjacency list.

    Parameters:
    adj_list (dict): The adjacency list representation of the graph.

    Returns:
    backbone_nodes (list): List of backbone nodes found in the graph.
    """
    degrees = {vertex: len(neighbors) for vertex, neighbors in adj_list.items()}

    leaves = [node for node, degree in degrees.items() if degree == 1]

    nodes = []
    for leaf in leaves:
        if degrees[leaf] == 1:
            degrees[leaf] = 0
            nodes.append(leaf)
            for neighbor in adj_list[leaf]:
                degrees[neighbor] -= 1

    for vertex in adj_list:
        if degrees[vertex] == 2:
            nodes.append(vertex)

    vertexes = list(adj_list.keys())
    backbone_nodes = [x for x in vertexes if x not in nodes]


    return backbone_nodes
```

Алгоритм розмітки суми для графів-гусеней

```python
def mark_caterpillar_graph(edges):
    """
    Marks the vertices of a caterpillar graph based on the given edges.

    Parameters:
    edges (list of tuples): List of edges in the form of tuples (u, v), where u and v are vertices.

    Returns:
    dict: Dictionary with vertices as keys and their corresponding labels as values.
    """
    if not edges:
        raise ValueError("The edge list is empty")

    adj_list = defaultdict(list)

    for u, v in edges:
        adj_list[u].append(v)
        adj_list[v].append(u)

    backbone = find_caterpillar_backbone(adj_list)

    if not backbone:
        raise ValueError("Задані ребра не формують граф-гусінь")

    label = {}
    current_label = 2
    for i, node in enumerate(backbone):
        if node not in label:
            label[node] = current_label
            current_label += 1

        sum_of_adj_labels = label[node]
        for neighbor in adj_list[node]:
            if neighbor not in label:
                label[neighbor] = current_label
                current_label += sum_of_adj_labels

    largest_neighbors = sorted(label.items(), reverse=True)[:2]
    label[max(label.keys()) + 1] = largest_neighbors[0][1] + largest_neighbors[1][1]

    return label
```

Алгоритм розмітки суми для графів-гусеней

```python
def draw_graph(nodes, edges):
    """
    This function draws a graph using NetworkX library.

    Parameters:
    nodes (dict): Dictionary where keys represent node names and values represent node numbers.
    edges (list of tuples): List of edges in the form of tuples (u, v), where u and v are vertices.

    Returns:
    No explicit return value. The function displays the graph.
    """
    G = nx.Graph()

    for node, number in nodes.items():
        G.add_node(node, number=number)

    G.add_edges_from(edges)

    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=False, node_size=2000, node_color='skyblue', font_size=16)

    labels = nx.get_node_attributes(G, 'number')
    nx.draw_networkx_labels(G, pos, labels, font_size=12)

    plt.title("Граф")
    plt.axis('off')
    plt.show()
```

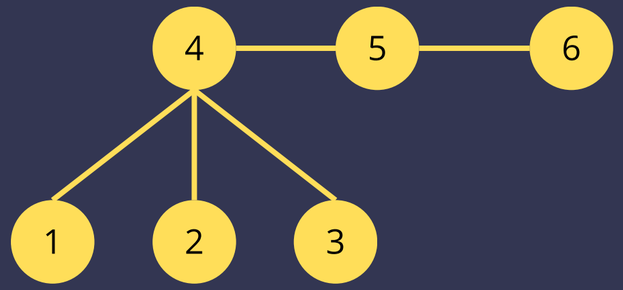
Алгоритм розмітки суми для графів-гусеней

```
def sum_mark_graph():
    cntn = True
    while cntn:
        seq = input_to_number_list('Введіть послідовність Прюфера (через кому): ')
        while not is_prufer_sequence(seq):
            print('Послідовність, яку Ви ввели, не є послідовністю Прюфера')
            seq = input_to_number_list('Введіть послідовність Прюфера (через кому): ')
        try:
            edges = prufer_sequence_to_edges(seq)
            labels = mark_caterpillar_graph(edges)
            draw_graph(labels, edges)
        except Exception as e:
            print(e)
            choice = input('Запустити алгоритм ще раз? (Y/y, якщо так і будь-що інше, якщо ні):')
            if choice.lower() != 'y':
                cntn = False
```

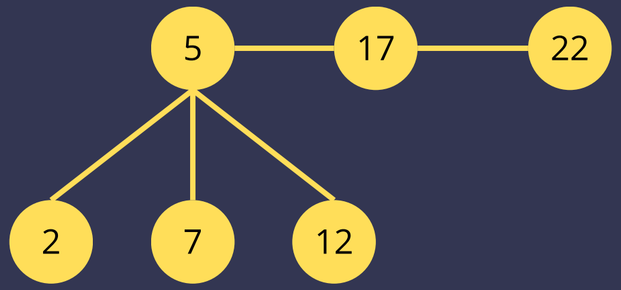
Алгоритм розмітки суми для графів-гусеней

Для послідовності Прюфера: 4,4,4,5

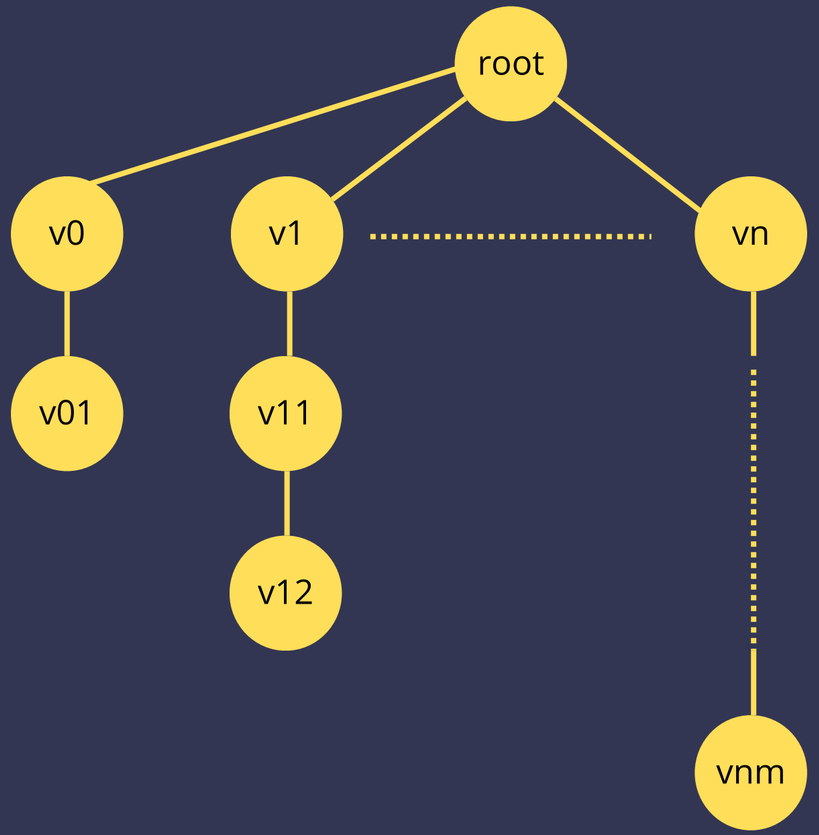
Граф-гусінь заданий послідовністю Прюфера:

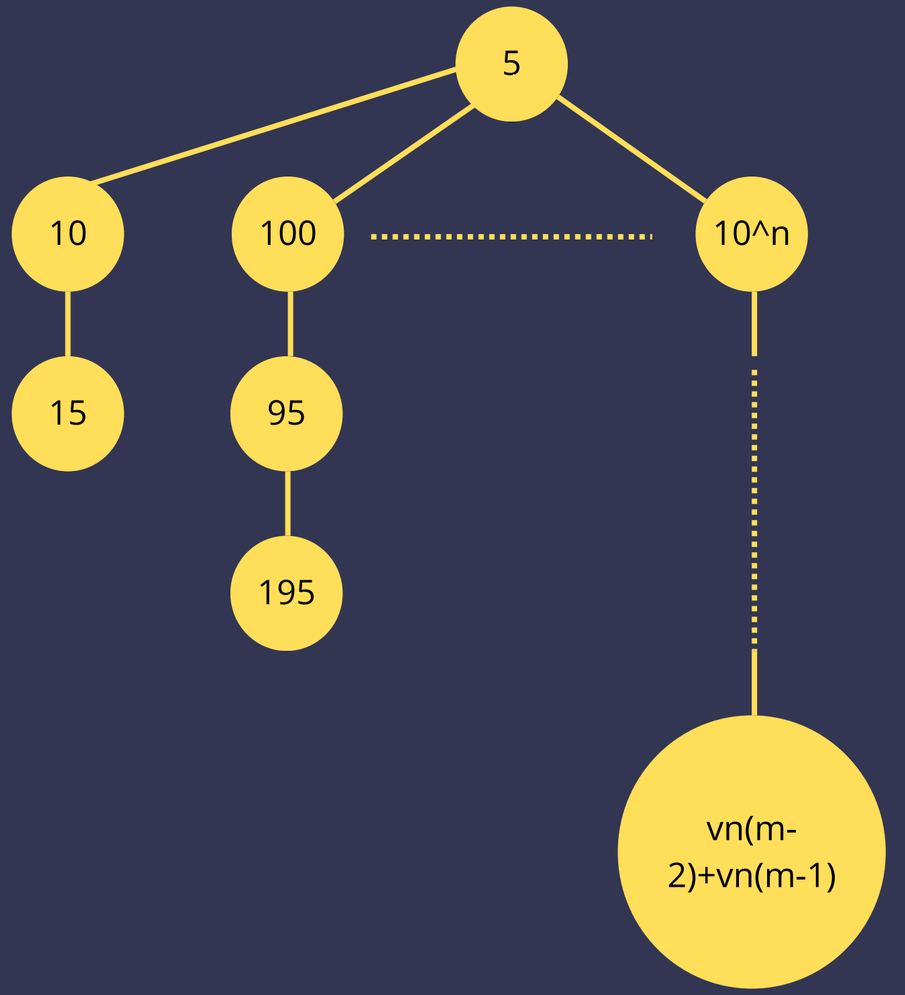


Маркування суми для заданого графа-гусіня:

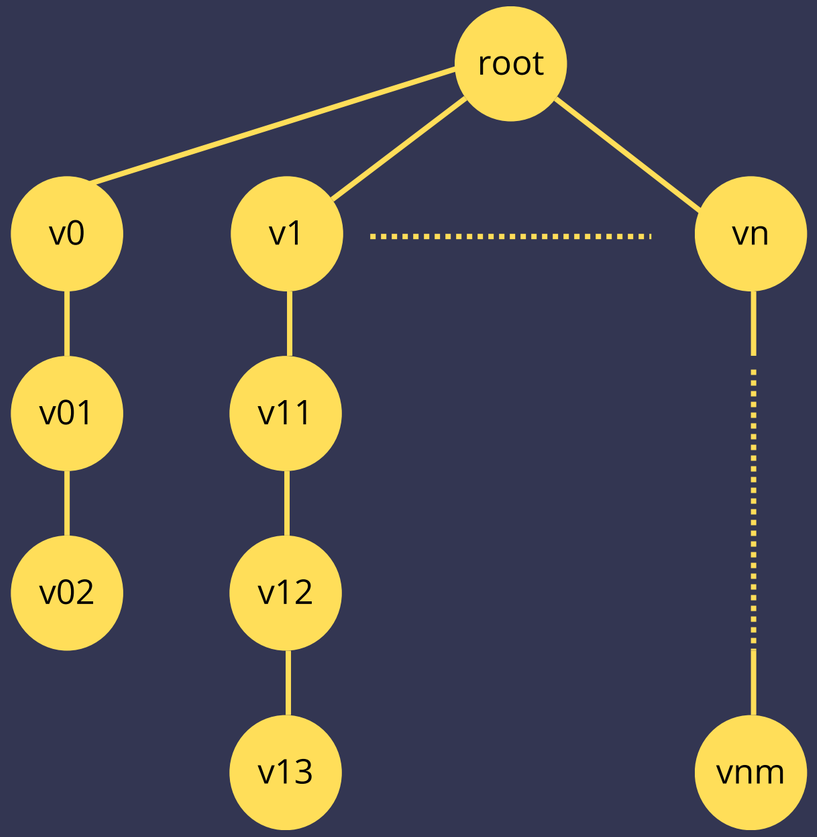


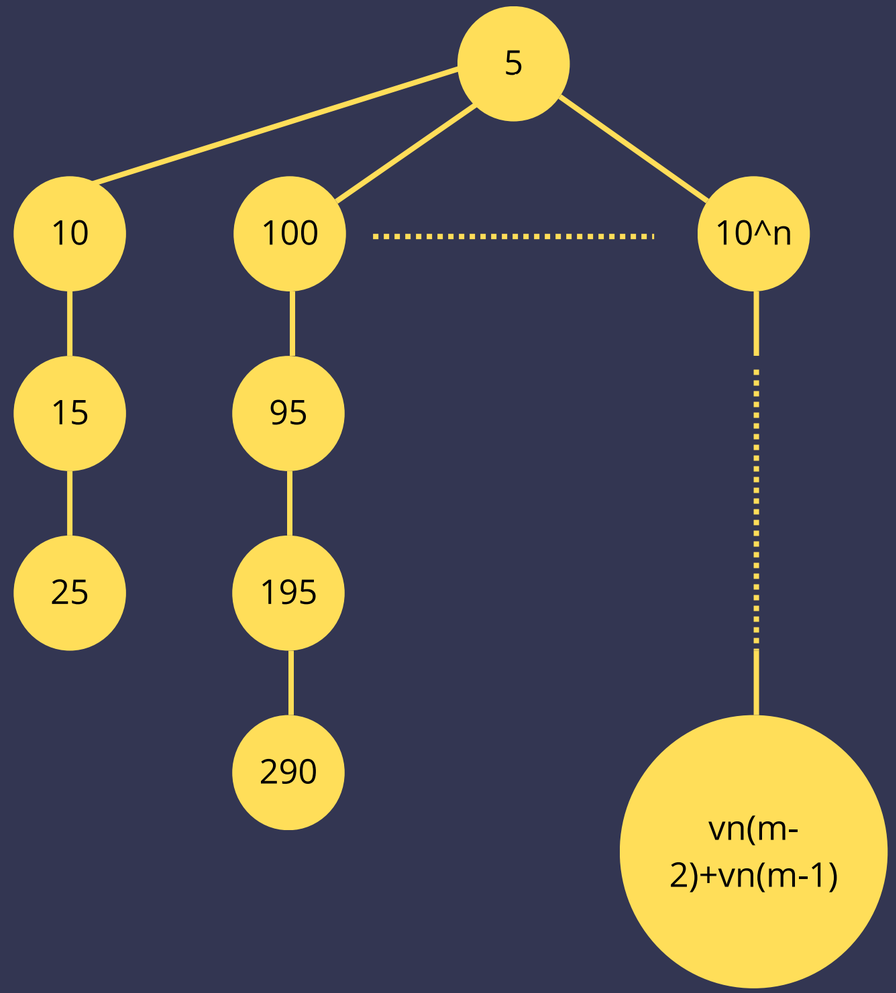
Загальна розмітка різниці для оливкових дерев





Загальна розмітка різниці для графів-павуків





Висновки

1. Досліджено основні властивості графів сум і різницевих графів.
2. Досліджено створення маркування суми для наступних родин графів: повних графів, дерев (зірки, бізірки, графи-гусені).
3. Досліджено створення маркування різниці для наступних родин графів: повних графів, дерев (зірки, бізірки, графи-гусені, оливкові дерева, графи-павуки).
4. Реалізовано алгоритм для маркування суми графів-гусеней.
5. Створено загальну розмітку різниці для оливкових дерев.
6. Створено загальну розмітку різниці для графів-павуків.

Список використаних джерел

1. Harary, F. *Sum graphs and difference graphs*. Congressus Numerantium, volume 72, 1990, pp. 101–108
2. Douglas B. West, *Introduction in Graph Theory*, Pearson Education Inc., 2001, 589 p.
3. Simon Schierreich, *Sum graphs*, [Електронний ресурс], https://dspace.cvut.cz/bitstream/handle/10467/90024/F8-DP-2020-Schierreich-Simon-thesis.pdf?sequence=-1\&isAllowed=y.
4. Bergstrand, D.; Harary, F.; et al. *The Sum Number of a Complete Graph*. Bulletin of the Malaysian Mathematical Sciences Society, volume 12, 1989: pp. 25–28.
5. Ellingham, M. N. *Sum graphs from trees*. Ars Combinatorica, volume 35, 1993: pp. 335–349
6. M. A. Seoud, M. M. Farid, M. Anwar, Some Difference Graphs, [Електронний ресурс], https://arxiv.org/pdf/2209.07317v1

Дякую за увагу!