Lampiran 1. Pewarnaan dengan Pemrograman

1. a. Pewarnaan pada graf $L_1 \rhd P_2$

```
# Langkah-langkah yang digunakan, yaitu:
1. Membuat simpul dan sisi.
2. Memastikan bahwa titik yang bertetangga tidak memiliki warna yang
   sama serta sisi yang terinduksi adalah ganjil.
3. Menambahkan aturan manual agar dua sisi yang bertetangga tidak sama.
4. Menampilkan hasil yang didapatkan dengan beberapa kali percobaan.
5. Menampilkan gambar dari hasil yang didapatkan.
from itertools import permutations, product
import networkx as nx
import matplotlib.pyplot as plt

class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                     [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []
        self.color = {}

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self.E.add((f'x_{i},1', f'x_{i+1},1'))
                self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self, coloring):
        for u, v in self.E:
            if u in coloring and v in coloring:
                if coloring[u] == coloring[v]:
                    return False
                if abs(coloring[u] - coloring[v]) % 2 == 0:
                    return False
```

```
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        valid_solutions = []
        total_checked = 0

        for coloring in product(self.colors, repeat=len(self.nodes)):
            total_checked += 1
            coloring_dict = dict(zip(self.nodes, coloring))
            if                self.is_valid_coloring(coloring_dict)                and
self.check_custom_constraints(coloring_dict):
                valid_solutions.append(coloring_dict)
                print(f"\nSolusi ke-{len(valid_solutions)}:")
                self.color = coloring_dict
                self.print_solution(coloring_dict)
                if len(valid_solutions) >= max_solutions:
                    break

        if not valid_solutions:
            print(f"Tidak ada pewarnaan valid dari {total_checked}
kemungkinan yang diperiksa.")

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
```

```
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        # Posisi simpul: x_(i,j) di atas, y_(i,j) di bawah
        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        # Label simpul: nilai warna
        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}

        # Warna simpul berdasarkan pewarnaan
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap,
node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels,
font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        # Label sisi = selisih warna
        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_color='darkred', font_size=10)

        plt.title('')
        plt.axis('off')
        plt.tight_layout()
        plt.show()

# Buat objek untuk n = 1
gc = GraphColoring(n=1, m=2, colors=[1, 2, 3, 4])

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_1,2'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_1,2'))

# Jalankan
gc.solve(max_solutions=10)
```

Output

```
Solusi ke-2:
Solusi ditemukan:
x_1,1 = 4
x_1,2 = 3
y_1,1 = 1
y_1,2 = 2

Selisih warna sisi:
x_1,1 - x_1,2 : |4 - 3| = 1
x_1,1 - y_1,1 : |4 - 1| = 3
y_1,1 - y_1,2 : |1 - 2| = 1
```

```
Solusi ke-1:
Solusi ditemukan:
x_1,1 = 1
x_1,2 = 2
y_1,1 = 4
y_1,2 = 3

Selisih warna sisi:
x_1,1 - x_1,2 : |1 - 2| = 1
x_1,1 - y_1,1 : |1 - 4| = 3
y_1,1 - y_1,2 : |4 - 3| = 1
```

1. b. Pewarnaan pada graf $L_1 \rhd P_m, m \geq 3$

```
from itertools import permutations, product
import networkx as nx
import matplotlib.pyplot as plt

class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                    [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []
        self.color = {}

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self.E.add((f'x_{i},1', f'x_{i+1},1'))
                self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
```

```
                self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self, coloring):
        for u, v in self.E:
            if u in coloring and v in coloring:
                if coloring[u] == coloring[v]:
                    return False
                if abs(coloring[u] - coloring[v]) % 2 == 0:
                    return False
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        valid_solutions = []
        total_checked = 0

        for coloring in product(self.colors, repeat=len(self.nodes)):
            total_checked += 1
            coloring_dict = dict(zip(self.nodes, coloring))
            if self.is_valid_coloring(coloring_dict) and
self.check_custom_constraints(coloring_dict):
                valid_solutions.append(coloring_dict)
                print(f"\nSolusi ke-{len(valid_solutions)}:")
                self.color = coloring_dict
                self.print_solution(coloring_dict)
```

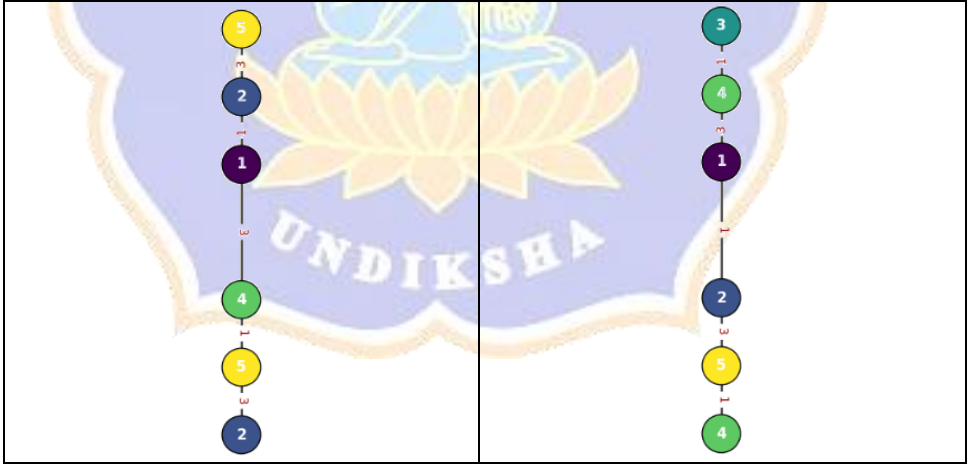
```
            if len(valid_solutions) >= max_solutions:
                break

        if not valid_solutions:
            print(f"Tidak ada pewarnaan valid dari {total_checked} kemungkinan yang diperiksa.")

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        # Posisi simpul: x_(i,j) di atas, y_(i,j) di bawah
        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        # Label simpul: nilai warna
        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}

        # Warna simpul berdasarkan pewarnaan
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap, node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels, font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        # Label sisi = selisih warna
        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='darkred', font_size=10)

        plt.title('')
        plt.axis('off')
```

```
        plt.tight_layout()
        plt.show()

# Buat objek untuk n = 1
gc = GraphColoring(n=1, m=3, colors=[1, 2, 3, 4, 5])

# Aturan khusus: label sisi yang bertetangga tidak boleh sama
gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_1,2'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_1,2'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,2', 'x_1,3'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,2', 'y_1,3'))

# Jalankan
gc.solve(max_solutions=10)
```

Output

```
Solusi ke-1:
Solusi ditemukan:
x_1,1 = 1
x_1,2 = 2
x_1,3 = 5
y_1,1 = 4
y_1,2 = 5
y_1,3 = 2

Selisih warna sisi:
x_1,1 - x_1,2 : |1 - 2| = 1
x_1,1 - y_1,1 : |1 - 4| = 3
x_1,2 - x_1,3 : |2 - 5| = 3
y_1,1 - y_1,2 : |4 - 5| = 1
y_1,2 - y_1,3 : |5 - 2| = 3
```

```
Solusi ditemukan:
x_1,1 = 1
x_1,2 = 4
x_1,3 = 3
y_1,1 = 2
y_1,2 = 5
y_1,3 = 4

Selisih warna sisi:
x_1,1 - x_1,2 : |1 - 4| = 3
x_1,1 - y_1,1 : |1 - 2| = 1
x_1,2 - x_1,3 : |4 - 3| = 1
y_1,1 - y_1,2 : |2 - 5| = 3
y_1,2 - y_1,3 : |5 - 4| = 1
```



1. c. Pewarnaan pada graf $L_2 \rhd P_m, m \geq 2$

```
from itertools import permutations, product
import networkx as nx
import matplotlib.pyplot as plt

class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
```

```
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []
        self.color = {}

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self.E.add((f'x_{i},1', f'x_{i+1},1'))
                self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self, coloring):
        for u, v in self.E:
            if u in coloring and v in coloring:
                if coloring[u] == coloring[v]:
                    return False
                if abs(coloring[u] - coloring[v]) % 2 == 0:
                    return False
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
```

```
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        valid_solutions = []
        total_checked = 0

        for coloring in product(self.colors, repeat=len(self.nodes)):
            total_checked += 1
            coloring_dict = dict(zip(self.nodes, coloring))
            if                self.is_valid_coloring(coloring_dict)                and
self.check_custom_constraints(coloring_dict):
                valid_solutions.append(coloring_dict)
                print(f"\nSolusi ke-{len(valid_solutions)}:")
                self.color = coloring_dict
                self.print_solution(coloring_dict)
                if len(valid_solutions) >= max_solutions:
                    break

        if not valid_solutions:
            print(f"Tidak ada pewarnaan valid dari {total_checked}
kemungkinan yang diperiksa.")

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        # Posisi simpul: x_(i,j) di atas, y_(i,j) di bawah
        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        # Label simpul: nilai warna
        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}

        # Warna simpul berdasarkan pewarnaan
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis
```

```
        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap, node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels, font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        # Label sisi = selisih warna
        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='darkred', font_size=10)

        plt.title('')
        plt.axis('off')
        plt.tight_layout()
        plt.show()

# Buat objek untuk n = 2
gc = GraphColoring(n=2, m=2, colors=[1, 2, 3, 4, 5, 6, 7])

# Aturan khusus: label sisi yang bertetangga tidak boleh sama
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_1,1', 'y_2,1'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_1,1', 'y_2,1'))

# Jalankan
gc.solve(max_solutions=10)
```

Output

```
Solusi ditemukan:
x_1,1 = 1
x_1,2 = 4
x_2,1 = 6
x_2,2 = 3
y_1,1 = 2
y_1,2 = 5
y_2,1 = 7
y_2,2 = 4

Selisih warna sisi:
x_1,1 - x_1,2 : |1 - 4| = 3
x_1,1 - x_2,1 : |1 - 6| = 5
x_1,1 - y_1,1 : |1 - 2| = 1
x_2,1 - x_2,2 : |6 - 3| = 3
x_2,1 - y_2,1 : |6 - 7| = 1
y_1,1 - y_1,2 : |2 - 5| = 3
y_1,1 - y_2,1 : |2 - 7| = 5
y_2,1 - y_2,2 : |7 - 4| = 3
```

1. d. Pewarnaan pada graf $L_3 \rhd P_m, m \geq 2$

```
import matplotlib.pyplot as plt
import networkx as nx

class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                     [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self.E.add((f'x_{i},1', f'x_{i+1},1'))
                self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_partial_valid(self, coloring, node, color):
        for u, v in self.E:
            if node in (u, v):
                other = v if node == u else u
                if other in coloring:
                    if coloring[other] == color:
```

```
                    return False
                if abs(coloring[other] - color) % 2 == 0:
                    return False
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        self.found = 0
        self._backtrack({}, 0, max_solutions)
        if self.found == 0:
            print("Tidak ada pewarnaan valid ditemukan.")

    def _backtrack(self, coloring, idx, max_solutions):
        if self.found >= max_solutions:
            return
        if idx == len(self.nodes):
            if self.check_custom_constraints(coloring):
                self.print_solution(coloring.copy())
                self.found += 1
            return
        node = self.nodes[idx]
        for color in self.colors:
            if self.is_partial_valid(coloring, node, color):
                coloring[node] = color
                self._backtrack(coloring, idx + 1, max_solutions)
                del coloring[node]

    def add_constraint(self, edge1, edge2):
```

```
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap,
node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels,
font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_color='darkred', font_size=10)

        plt.title("")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

gc = GraphColoring(n=3, m=2, colors=[1, 2, 3, 4, 5, 6, 7, 8])

# Aturan khusus: label sisi yang bertetangga tidak boleh sama
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_2,1', 'x_3,1'))
```

```
gc.add_constraint(('y_1,1', 'y_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('y_2,1', 'y_3,1'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('y_2,1', 'y_3,1'))

gc.add_constraint(('x_1,1', 'x_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('y_1,1', 'y_2,1'), ('y_2,1', 'y_3,1'))

gc.solve(max_solutions=10)
```

Output

```
Solusi ditemukan:
x_1,1 = 2
x_1,2 = 5
x_2,1 = 1
x_2,2 = 4
x_3,1 = 6
x_3,2 = 5
y_1,1 = 7
y_1,2 = 4
y_2,1 = 8
y_2,2 = 5
y_3,1 = 3
y_3,2 = 2

Selisih warna sisi:
x_1,1 - x_1,2 : |2 - 5| = 3
x_1,1 - x_2,1 : |2 - 1| = 1
x_1,1 - y_1,1 : |2 - 7| = 5
x_2,1 - x_2,2 : |1 - 4| = 3
x_2,1 - x_3,1 : |1 - 6| = 5
x_2,1 - y_2,1 : |1 - 8| = 7
x_3,1 - x_3,2 : |6 - 5| = 1
x_3,1 - y_3,1 : |6 - 3| = 3
y_1,1 - y_1,2 : |7 - 4| = 3
y_1,1 - y_2,1 : |7 - 8| = 1
y_2,1 - y_2,2 : |8 - 5| = 3
y_2,1 - y_3,1 : |8 - 3| = 5
y_3,1 - y_3,2 : |3 - 2| = 1
```

1. e. Pewarnaan pada graf $L_4 \rhd P_m, m \geq 2$

```
import matplotlib.pyplot as plt
import networkx as nx

import time
```

```
class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                     [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []
        self.adj = {node: set() for node in self.nodes}
        self.edge_labels = dict()

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self._add_edge(f'x_{i},1', f'x_{i+1},1')
                self._add_edge(f'y_{i},1', f'y_{i+1},1')

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self._add_edge(f'x_{i},{j}', f'x_{i},{j+1}')
                self._add_edge(f'y_{i},{j}', f'y_{i},{j+1}')

        for i in range(1, self.n + 1):
            self._add_edge(f'x_{i},1', f'y_{i},1')

    def _add_edge(self, u, v):
        self.E.add((u, v))
        self.adj[u].add(v)
        self.adj[v].add(u)

    def is_partial_valid(self, coloring, node, color):
        for neighbor in self.adj[node]:
            if neighbor in coloring:
                if coloring[neighbor] == color:
                    return False
                if abs(coloring[neighbor] - color) % 2 == 0:
                    return False

                edge = tuple(sorted((node, neighbor)))
                label = abs(coloring[neighbor] - color)
                if edge in self.edge_labels:
                    return False
                for other_edge in self.edge_labels:
                    if node in other_edge or neighbor in other_edge:
                        if self.edge_labels[other_edge] == label:
                            return False
```

```
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        self.found = 0
        start_time = time.time()
        self._backtrack({}, 0, max_solutions)
        if self.found == 0:
            print("Tidak ada pewarnaan valid ditemukan.")

    def _backtrack(self, coloring, idx, max_solutions):
        if self.found >= max_solutions:
            return
        if idx == len(self.nodes):
            if self.check_custom_constraints(coloring):
                self.print_solution(coloring.copy())
                self.found += 1
            return
        node = self.nodes[idx]
        for color in self.colors:
            if self.is_partial_valid(coloring, node, color):
                coloring[node] = color
                for neighbor in self.adj[node]:
                    if neighbor in coloring:
                        edge = tuple(sorted((node, neighbor)))
                        self.edge_labels[edge]      =      abs(coloring[node]      -
coloring[neighbor])
                self._backtrack(coloring, idx + 1, max_solutions)
```

```
            for neighbor in self.adj[node]:
                if neighbor in coloring:
                    edge = tuple(sorted((node, neighbor)))
                    if edge in self.edge_labels:
                        del self.edge_labels[edge]
            del coloring[node]

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap,
node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels,
font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_color='darkred', font_size=10)

        plt.title("")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

gc = GraphColoring(n=4, m=2, colors=[1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
# Aturan khusus: label sisi yang bertetangga tidak boleh sama
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'x_4,2'), ('x_4,1', 'y_4,1'))
gc.add_constraint(('x_4,1', 'x_4,2'), ('x_3,1', 'x_4,1'))

gc.add_constraint(('y_1,1', 'y_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('y_4,1', 'y_4,2'), ('x_4,1', 'y_4,1'))
gc.add_constraint(('y_4,1', 'y_4,2'), ('y_3,1', 'y_4,1'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('y_3,1', 'y_4,1'))

gc.add_constraint(('x_1,1', 'x_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('y_1,1', 'y_2,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_2,1', 'x_3,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('y_2,1', 'y_3,1'), ('y_3,1', 'y_4,1'))

gc.solve(max_solutions=10)
```

Output

```
Solusi ditemukan:
x_1,1 = 3
x_1,2 = 2
x_2,1 = 8
x_2,2 = 5
x_3,1 = 9
x_3,2 = 6
x_4,1 = 4
x_4,2 = 3
y_1,1 = 6
y_1,2 = 5
y_2,1 = 1
y_2,2 = 4
y_3,1 = 2
y_3,2 = 5
y_4,1 = 7
y_4,2 = 6
```

```
Selisih warna sisi:
x_1,1 - x_1,2 : |3 - 2| = 1
x_1,1 - x_2,1 : |3 - 8| = 5
x_1,1 - y_1,1 : |3 - 6| = 3
x_2,1 - x_2,2 : |8 - 5| = 3
x_2,1 - x_3,1 : |8 - 9| = 1
x_2,1 - y_2,1 : |8 - 1| = 7
x_3,1 - x_3,2 : |9 - 6| = 3
x_3,1 - x_4,1 : |9 - 4| = 5
x_3,1 - y_3,1 : |9 - 2| = 7
x_4,1 - x_4,2 : |4 - 3| = 1
x_4,1 - y_4,1 : |4 - 7| = 3
y_1,1 - y_1,2 : |6 - 5| = 1
y_1,1 - y_2,1 : |6 - 1| = 5
y_2,1 - y_2,2 : |1 - 4| = 3
y_2,1 - y_3,1 : |1 - 2| = 1
y_3,1 - y_3,2 : |2 - 5| = 3
y_3,1 - y_4,1 : |2 - 7| = 5
y_4,1 - y_4,2 : |7 - 6| = 1
```

1. f. Pewarnaan pada graf $L_5 \rhd P_m, m \geq 2$

```
import matplotlib.pyplot as plt
import networkx as nx

import time

class GraphColoring:
    def __init__(self, n, m, colors):
        self.n = n
        self.m = m
        self.colors = colors
        self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
+ \
                     [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
        self.E = set()
        self.custom_edge_constraints = []
        self.adj = {node: set() for node in self.nodes}
        self.edge_labels = dict()

        for j in range(1, self.m + 1):
            for i in range(1, self.n):
                self._add_edge(f'x_{i},1', f'x_{i+1},1')
                self._add_edge(f'y_{i},1', f'y_{i+1},1')

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self._add_edge(f'x_{i},{j}', f'x_{i},{j+1}')
                self._add_edge(f'y_{i},{j}', f'y_{i},{j+1}')

        for i in range(1, self.n + 1):
```

```
            self._add_edge(f'x_{i},1', f'y_{i},1')

    def _add_edge(self, u, v):
        self.E.add((u, v))
        self.adj[u].add(v)
        self.adj[v].add(u)

    def is_partial_valid(self, coloring, node, color):
        for neighbor in self.adj[node]:
            if neighbor in coloring:
                if coloring[neighbor] == color:
                    return False
                if abs(coloring[neighbor] - color) % 2 == 0:
                    return False

                edge = tuple(sorted((node, neighbor)))
                label = abs(coloring[neighbor] - color)
                if edge in self.edge_labels:
                    return False
                for other_edge in self.edge_labels:
                    if node in other_edge or neighbor in other_edge:
                        if self.edge_labels[other_edge] == label:
                            return False
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2, v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} - {coloring[v]}| = {abs(coloring[u]
- coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        self.found = 0
```

```
        start_time = time.time()
        self._backtrack({}, 0, max_solutions)
        if self.found == 0:
            print("Tidak ada pewarnaan valid ditemukan.")

    def _backtrack(self, coloring, idx, max_solutions):
        if self.found >= max_solutions:
            return
        if idx == len(self.nodes):
            if self.check_custom_constraints(coloring):
                self.print_solution(coloring.copy())
                self.found += 1
            return
        node = self.nodes[idx]
        for color in self.colors:
            if self.is_partial_valid(coloring, node, color):
                coloring[node] = color
                for neighbor in self.adj[node]:
                    if neighbor in coloring:
                        edge = tuple(sorted((node, neighbor)))
                        self.edge_labels[edge] = abs(coloring[node] -
coloring[neighbor])
                self._backtrack(coloring, idx + 1, max_solutions)
                for neighbor in self.adj[node]:
                    if neighbor in coloring:
                        edge = tuple(sorted((node, neighbor)))
                        if edge in self.edge_labels:
                            del self.edge_labels[edge]
                del coloring[node]

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        pos = {}
        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}
        values = [self.color.get(v, 0.25) for v in G.nodes]
```

```
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap, node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels, font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='darkred', font_size=10)

        plt.title("")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

gc = GraphColoring(n=5, m=2, colors=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Aturan khusus: label sisi yang bertetangga tidak boleh sama
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'x_2,2'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'x_3,2'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'x_4,2'), ('x_4,1', 'y_4,1'))
gc.add_constraint(('x_4,1', 'x_4,2'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'x_4,2'), ('x_4,1', 'x_5,1'))
gc.add_constraint(('x_5,1', 'x_5,2'), ('x_5,1', 'y_5,1'))
gc.add_constraint(('x_5,1', 'x_5,2'), ('x_4,1', 'x_5,1'))

gc.add_constraint(('y_1,1', 'y_1,2'), ('x_1,1', 'y_1,1'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('x_2,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('y_2,1', 'y_2,2'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('x_3,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('y_3,1', 'y_3,2'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('y_4,1', 'y_4,2'), ('x_4,1', 'y_4,1'))
gc.add_constraint(('y_4,1', 'y_4,2'), ('y_3,1', 'y_4,1'))
```

```
gc.add_constraint(('y_4,1', 'y_4,2'), ('y_4,1', 'y_5,1'))
gc.add_constraint(('y_5,1', 'y_5,2'), ('x_5,1', 'y_5,1'))
gc.add_constraint(('y_5,1', 'y_5,2'), ('y_4,1', 'y_5,1'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_1,1', 'x_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_1,1', 'y_2,1'))
gc.add_constraint(('x_2,1', 'y_2,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('x_3,1', 'y_3,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('x_4,1', 'x_5,1'))
gc.add_constraint(('x_4,1', 'y_4,1'), ('y_4,1', 'y_5,1'))
gc.add_constraint(('x_5,1', 'y_5,1'), ('x_4,1', 'x_5,1'))
gc.add_constraint(('x_5,1', 'y_5,1'), ('y_4,1', 'y_5,1'))

gc.add_constraint(('x_1,1', 'x_2,1'), ('x_2,1', 'x_3,1'))
gc.add_constraint(('y_1,1', 'y_2,1'), ('y_2,1', 'y_3,1'))
gc.add_constraint(('x_2,1', 'x_3,1'), ('x_3,1', 'x_4,1'))
gc.add_constraint(('y_2,1', 'y_3,1'), ('y_3,1', 'y_4,1'))
gc.add_constraint(('x_3,1', 'x_4,1'), ('x_4,1', 'x_5,1'))
gc.add_constraint(('y_3,1', 'y_4,1'), ('y_4,1', 'y_5,1'))

gc.solve(max_solutions=16)
```

Output

```
Solusi ditemukan:
x_1,1 = 4
x_1,2 = 3
x_2,1 = 9
x_2,2 = 6
x_3,1 = 10
x_3,2 = 7
x_4,1 = 3
x_4,2 = 4
x_5,1 = 6
x_5,2 = 1
y_1,1 = 7
y_1,2 = 8
y_2,1 = 2
y_2,2 = 5
y_3,1 = 1
y_3,2 = 6
y_4,1 = 8
y_4,2 = 9
y_5,1 = 5
y_5,2 = 10

Selisih warna sisi:
x_1,1 - x_1,2 : |4 - 3| = 1
x_1,1 - x_2,1 : |4 - 9| = 5
x_1,1 - y_1,1 : |4 - 7| = 3
x_2,1 - x_2,2 : |9 - 6| = 3
x_2,1 - x_3,1 : |9 - 10| = 1
x_2,1 - y_2,1 : |9 - 2| = 7
x_3,1 - x_3,2 : |10 - 7| = 3
x_3,1 - x_4,1 : |10 - 3| = 7
x_3,1 - y_3,1 : |10 - 1| = 9
x_4,1 - x_4,2 : |3 - 4| = 1
x_4,1 - x_5,1 : |3 - 6| = 3
x_4,1 - y_4,1 : |3 - 8| = 5
x_5,1 - x_5,2 : |6 - 1| = 5
x_5,1 - y_5,1 : |6 - 5| = 1
y_1,1 - y_1,2 : |7 - 8| = 1
y_1,1 - y_2,1 : |7 - 2| = 5
y_2,1 - y_2,2 : |2 - 5| = 3
y_2,1 - y_3,1 : |2 - 1| = 1
y_3,1 - y_3,2 : |1 - 6| = 5
y_3,1 - y_4,1 : |1 - 8| = 7
y_4,1 - y_4,2 : |8 - 9| = 1
y_4,1 - y_5,1 : |8 - 5| = 3
y_5,1 - y_5,2 : |5 - 10| = 5
```

1. g. Pewarnaan pada graf $L_n \rhd P_m, n \geq 6, m \geq 2$

```
from math import e
import matplotlib.pyplot as plt
```

```
import networkx as nx

class Graph:
    def __init__(self, n, m):
        self.n = n  # Jumlah blok L_n
        self.m = m  # Jumlah simpul pada setiap P_m
        self.V = set()  # Himpunan simpul
        self.E = set()  # Himpunan sisi
        self.color = {}  # Warna untuk setiap simpul

        self._generate_graph()

    def _generate_graph(self):
        """
        Membuat graf berdasarkan aturan blok comb.
        """
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                self.V.add(f'x_{i},{j}')
                self.V.add(f'y_{i},{j}')

        for i in range(1, self.n):
            self.E.add((f'x_{i},1', f'x_{i+1},1'))
            self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def apply_coloring_rule(self):
        """
        Menerapkan aturan pewarnaan graceful ganjil berdasarkan indeks modular.
        """
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                # Pewarnaan x
                if i % 4 == 1 and j % 4 == 1:
                    self.color[f'x_{i},{j}'] = 8
                elif i % 4 == 1 and j % 4 == 2:
                    self.color[f'x_{i},{j}'] = 5
                elif i % 4 == 1 and j % 4 == 3:
                    self.color[f'x_{i},{j}'] = 6
                elif i % 4 == 1 and j % 4 == 0:
```

```
    self.color[f'x_{i},{j}'] = 9
elif i % 4 == 2 and j % 4 == 1:
    self.color[f'x_{i},{j}'] = 3
elif i % 4 == 2 and j % 4 == 2:
    self.color[f'x_{i},{j}'] = 6
elif i % 4 == 2 and j % 4 == 3:
    self.color[f'x_{i},{j}'] = 7
elif i % 4 == 2 and j % 4 == 0:
    self.color[f'x_{i},{j}'] = 4
elif i % 4 == 3 and j % 4 == 1:
    self.color[f'x_{i},{j}'] = 10
elif i % 4 == 3 and j % 4 == 2:
    self.color[f'x_{i},{j}'] = 7
elif i % 4 == 3 and j % 4 == 3:
    self.color[f'x_{i},{j}'] = 8
elif i % 4 == 3 and j % 4 == 0:
    self.color[f'x_{i},{j}'] = 11
elif i % 4 == 0 and j % 4 == 1:
    self.color[f'x_{i},{j}'] = 1
elif i % 4 == 0 and j % 4 == 2:
    self.color[f'x_{i},{j}'] = 4
elif i % 4 == 0 and j % 4 == 3:
    self.color[f'x_{i},{j}'] = 5
elif i % 4 == 0 and j % 4 == 0:
    self.color[f'x_{i},{j}'] = 2

# Pewarnaan y
if i % 4 == 1 and j % 4 == 1:
    self.color[f'y_{i},{j}'] = 9
elif i % 4 == 1 and j % 4 == 2:
    self.color[f'y_{i},{j}'] = 6
elif i % 4 == 1 and j % 4 == 3:
    self.color[f'y_{i},{j}'] = 5
elif i % 4 == 1 and j % 4 == 0:
    self.color[f'y_{i},{j}'] = 8
elif i % 4 == 2 and j % 4 == 1:
    self.color[f'y_{i},{j}'] = 4
elif i % 4 == 2 and j % 4 == 2:
    self.color[f'y_{i},{j}'] = 7
elif i % 4 == 2 and j % 4 == 3:
    self.color[f'y_{i},{j}'] = 6
elif i % 4 == 2 and j % 4 == 0:
    self.color[f'y_{i},{j}'] = 3
elif i % 4 == 3 and j % 4 == 1:
    self.color[f'y_{i},{j}'] = 11
elif i % 4 == 3 and j % 4 == 2:
    self.color[f'y_{i},{j}'] = 8
elif i % 4 == 3 and j % 4 == 3:
```

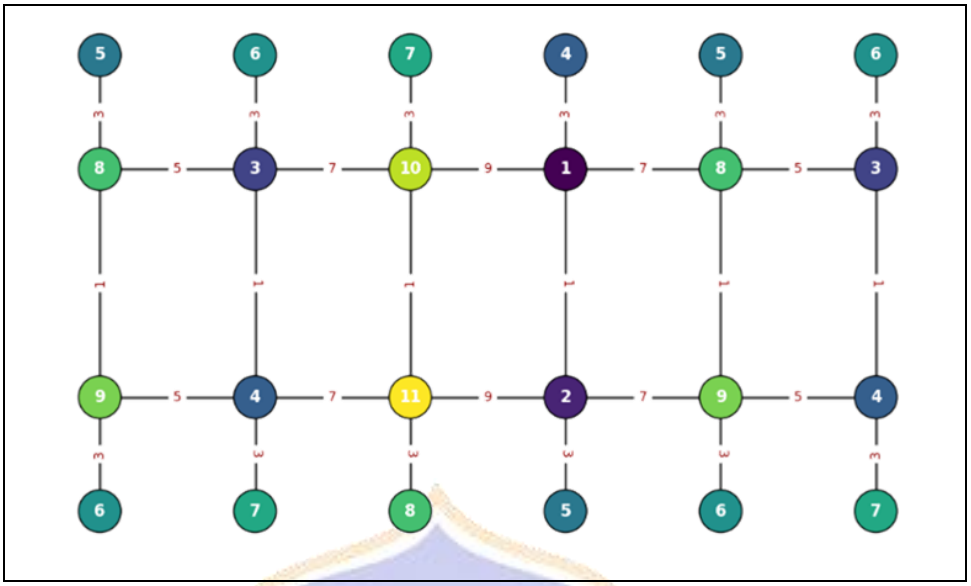
```
                self.color[f'y_{i},{j}'] = 7
            elif i % 4 == 3 and j % 4 == 0:
                self.color[f'y_{i},{j}'] = 10
            elif i % 4 == 0 and j % 4 == 1:
                self.color[f'y_{i},{j}'] = 2
            elif i % 4 == 0 and j % 4 == 2:
                self.color[f'y_{i},{j}'] = 5
            elif i % 4 == 0 and j % 4 == 3:
                self.color[f'y_{i},{j}'] = 4
            elif i % 4 == 0 and j % 4 == 0:
                self.color[f'y_{i},{j}'] = 1

    def display(self):
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} - {self.color[v]}| =
{abs(self.color[u] - self.color[v])}')
            else:
                print(f'{u} - {v}: Belum diwarnai')

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.V)
        G.add_edges_from(self.E)

        # Posisi simpul: x_(i,j) di atas, y_(i,j) di bawah
        pos = {}
        for node in self.V:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))
            x = i * 2
            y = j if typ == 'x' else -j
            pos[node] = (x, y)

        # Label simpul: nilai warna
        node_labels = {v: str(self.color.get(v, '?')) for v in G.nodes}

        # Warna simpul berdasarkan pewarnaan
        values = [self.color.get(v, 0.25) for v in G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
```

```
        nx.draw_networkx_nodes(G, pos, node_color=values, cmap=cmap, node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos, labels=node_labels, font_color='white', font_weight='bold')
        nx.draw_networkx_edges(G, pos, width=2, alpha=0.7)

        # Label sisi = selisih warna
        edge_labels = {}
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] = str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='darkred', font_size=10)

        plt.title('')
        plt.axis('off')
        plt.tight_layout()
        plt.show()

# Contoh eksekusi untuk n = 6, m = ...
graph = Graph(n=6, m=2)
graph.apply_coloring_rule()
graph.display()
graph.draw()
```

Output

```
Simpul dan Warnanya:
x_1,1: 8
x_1,2: 5
x_2,1: 3
x_2,2: 6
x_3,1: 10
x_3,2: 7
x_4,1: 1
x_4,2: 4
x_5,1: 8
x_5,2: 5
x_6,1: 3
x_6,2: 6
y_1,1: 9
y_1,2: 6
y_2,1: 4
y_2,2: 7
y_3,1: 11
y_3,2: 8
y_4,1: 2
y_4,2: 5
y_5,1: 9
y_5,2: 6
y_6,1: 4
y_6,2: 7
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |8 - 5| = 3
x_1,1 - x_2,1: |8 - 3| = 5
x_1,1 - y_1,1: |8 - 9| = 1
x_2,1 - x_2,2: |3 - 6| = 3
x_2,1 - x_3,1: |3 - 10| = 7
x_2,1 - y_2,1: |3 - 4| = 1
x_3,1 - x_3,2: |10 - 7| = 3
x_3,1 - x_4,1: |10 - 1| = 9
x_3,1 - y_3,1: |10 - 11| = 1
x_4,1 - x_4,2: |1 - 4| = 3
x_4,1 - x_5,1: |1 - 8| = 7
x_4,1 - y_4,1: |1 - 2| = 1
x_5,1 - x_5,2: |8 - 5| = 3
x_5,1 - x_6,1: |8 - 3| = 5
x_5,1 - y_5,1: |8 - 9| = 1
x_6,1 - x_6,2: |3 - 6| = 3
x_6,1 - y_6,1: |3 - 4| = 1
y_1,1 - y_1,2: |9 - 6| = 3
y_1,1 - y_2,1: |9 - 4| = 5
y_2,1 - y_2,2: |4 - 7| = 3
y_2,1 - y_3,1: |4 - 11| = 7
y_3,1 - y_3,2: |11 - 8| = 3
y_3,1 - y_4,1: |11 - 2| = 9
y_4,1 - y_4,2: |2 - 5| = 3
y_4,1 - y_5,1: |2 - 9| = 7
y_5,1 - y_5,2: |9 - 6| = 3
y_5,1 - y_6,1: |9 - 4| = 5
y_6,1 - y_6,2: |4 - 7| = 3
```



1. h. Pewarnaan pada graf $L_1 \rhd C_m$, m merupakan bilangan genap dan $m \geq 4$

Langkah-langkah yang digunakan, yaitu:
1. Membuat simpul dan sisi.
2. Memastikan bahwa titik yang bertetangga tidak memiliki warna yang sama serta sisi yang terinduksi adalah ganjil.
3. Menambahkan aturan manual agar dua sisi yang bertetangga tidak sama.
4. Menampilkan hasil yang didapatkan dengan beberapa kali percobaan.
5. Menampilkan gambar dari hasil yang didapatkan.

Kode	Output

from itertools import product
import networkx as nx
import matplotlib.pyplot as plt
import math
 	
class GraphColoring:
  def __init__(self, n, m, colors):
  self.n = n
  self.m = m
  self.colors = colors
  self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)] + \
  [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
  self.E = set()
  self.custom_edge_constraints = []
  self.color = {}
 	
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
	Solusi ditemukan:
	x_1,1 = 1
	x_1,2 = 2
	x_1,3 = 5
	x_1,4 = 4
	y_1,1 = 6
	y_1,2 = 3
	y_1,3 = 2
	y_1,4 = 5
	
	Selisih warna sisi:
	x_1,1 - x_1,2 : |1 - 2| = 1
	x_1,1 - x_1,4 : |1 - 4| = 3
	x_1,1 - y_1,1 : |1 - 6| = 5
	x_1,2 - x_1,3 : |2 - 5| = 3
	x_1,3 - x_1,4 : |5 - 4| = 1
	y_1,1 - y_1,2 : |6 - 3| = 3
	y_1,1 - y_1,4 : |6 - 5| = 1
	y_1,2 - y_1,3 : |3 - 2| = 1
	y_1,3 - y_1,4 : |2 - 5| = 3

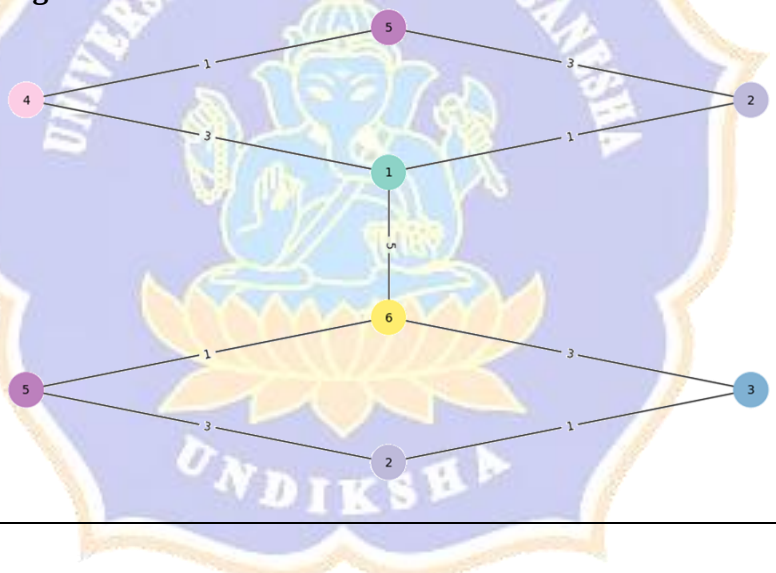
```
        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}',
f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}',
f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))
            self.E.add((f'x_{i},1', f'x_{i},{self.m}'))
            self.E.add((f'y_{i},1', f'y_{i},{self.m}'))

    def is_valid_coloring(self, coloring):
        for u, v in self.E:
            if u in coloring and v in coloring:
                if coloring[u] == coloring[v]:
                    return False
                if abs(coloring[u] - coloring[v]) % 2
== 0:
                    return False
        return True

    def check_custom_constraints(self, coloring):
        for (u1, v1), (u2, v2) in
self.custom_edge_constraints:
            if all(k in coloring for k in [u1, v1, u2,
v2]):
                d1 = abs(coloring[u1] - coloring[v1])
                d2 = abs(coloring[u2] - coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} -
{coloring[v]}| = {abs(coloring[u] -
coloring[v])}")

        self.draw(coloring)

    def draw(self, coloring):
```

```
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        # Posisi simpul dalam bentuk lingkaran
untuk setiap cycle
        pos = {}
        radius = 1
        angle_map_x = {
            1: 3 * math.pi / 2,
            2: 0,
            3: math.pi / 2,
            4: math.pi,
        }

        angle_map_y = {
            1: math.pi / 2,
            2: 0,
            3: 3 * math.pi / 2,
            4: math.pi,
        }

        for node in self.nodes:
            typ, idx = node.split('_')
            i, j = map(int, idx.split(','))

            # Pilih angle map berdasarkan jenis
simpul (x atau y)
            if self.m == 4:
                angle = angle_map_x[j] if typ == 'x'
else angle_map_y[j]
            else:
                angle = 2 * math.pi * (j - 1) / self.m

            cx = i * 4
            cy = 2 if typ == 'x' else -2

            x = cx + radius * math.cos(angle)
            y = cy + radius * math.sin(angle)
            pos[node] = (x, y)

        node_colors = [coloring[node] for node in
self.nodes]
        edge_labels = {(u, v): abs(coloring[u] -
coloring[v]) for u, v in G.edges()}

        plt.figure(figsize=(10, 6))
```

```
        node_labels = {node: str(coloring[node])
for node in self.nodes}
        nx.draw(G, pos, labels=node_labels,
node_color=node_colors, cmap=plt.cm.Set3,
node_size=800, font_size=10)
        nx.draw_networkx_edge_labels(G, pos,
edge_labels=edge_labels, font_color='black')

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

    def solve(self, max_solutions=1):
        valid_solutions = []
        total_checked = 0

        for coloring in product(self.colors,
repeat=len(self.nodes)):
            total_checked += 1
            coloring_dict = dict(zip(self.nodes,
coloring))
            if self.is_valid_coloring(coloring_dict)
and
self.check_custom_constraints(coloring_dict):
                valid_solutions.append(coloring_dict)
                self.print_solution(coloring_dict)
                if len(valid_solutions) >=
max_solutions:
                    break

        if not valid_solutions:
            print(f"Tidak ada pewarnaan valid dari
{total_checked} kemungkinan yang diperiksa.")

    def add_constraint(self, edge1, edge2):

self.custom_edge_constraints.append((edge1,
edge2))

# Buat objek untuk n = 1
gc = GraphColoring(n=1, m=4, colors=[1, 2, 3,
4, 5, 6])

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1',
'x_1,2'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,2',
'x_1,3'))
```

```
gc.add_constraint(('x_1,2', 'x_1,3'), ('x_1,3',
'x_1,4'))
gc.add_constraint(('x_1,3', 'x_1,4'), ('x_1,4',
'x_1,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1',
'x_1,4'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1',
'y_1,2'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,2',
'y_1,3'))
gc.add_constraint(('y_1,2', 'y_1,3'), ('y_1,3',
'y_1,4'))
gc.add_constraint(('y_1,3', 'y_1,4'), ('y_1,4',
'y_1,1'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1',
'y_1,4'))

gc.solve(max_solutions=1)
```

Output gambar



1. i. Pewarnaan pada graf $L_2 \rhd C_m$, m merupakan bilangan genap dan $m \geq 4$

Langkah-langkah yang digunakan, yaitu:

1. Membuat simpul dan sisi
2. Memastikan bahwa sisi yang terinduksi adalah ganjil
3. Mencari pewarnaan graceful ganjil untuk $x_{1,2}, \ldots, x_{1,m}$, $x_{2,2}, \ldots, x_{2,m}$, $y_{1,2}, \ldots, y_{1,m}$, dan $y_{2,2}, \ldots, y_{2,m}$
4. Gunakan aturan-aturan untuk $x_{1,2}$, yaitu sisi yang terinduksi harus ganjil, warnanya tidak boleh sama dengan $x_{1,1}, x_{2,1}$, dan $y_{1,1}$ karena titik-titik tersebut bertetangga dengan $x_{1,2}$

5. Gunakan aturan-aturan untuk $y_{1,2}$, yaitu sisi yang terinduksi harus ganjil, warnanya tidak boleh sama dengan $x_{1,1}, y_{2,1}$, dan $y_{1,1}$ karena titik-titik tersebut bertetangga dengan $y_{1,2}$
6. Untuk titik-titik selanjutnya gunakan aturan yang sama, yaitu sisi yang terinduksi harus ganjil dan titik-titik yang bertetangga tidak boleh memiliki warna yang sama

Kode	Output

class Graph:
  def __init__(self, n, m):
  self.n = n
  self.m = m
  self.V = set()
  self.E = set()
  self.color = {}
 	
  # Generate graf
  self._generate_graph()
 	
  def _generate_graph(self):
  """
  Membuat graf berdasarkan aturan yang telah dijelaskan.
  """
  # Generate simpul x_(i,j) dan y_(i,j)
  for i in range(1, self.n + 1):
  for j in range(1, self.m + 1):
  self.V.add(f'x_{i},{j}')
  self.V.add(f'y_{i},{j}')
 	
  # Generate sisi horizontal antara x_(i,1) dan x_(i+1,1), serta y_(i,1) dan y_(i+1,1)
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
 	
  # Generate sisi vertikal antara x_(i,j) dan x_(i,j+1), serta y_(i,j) dan y_(i,j+1)
  for i in range(1, self.n + 1):
  for j in range(1, self.m):
  self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
  self.E.add((f'y_{i},{j}', f'y_{i},{j+1}'))
 	
  # Generate sisi langsung dari x_(i,1) ke x_(i,m) dan y_(i,1) ke y_(i,m)
  for i in range(1, self.n + 1):
  self.E.add((f'x_{i},1', f'x_{i},{self.m}'))
	Pewarnaan ditemukan!
	Simpul dan Warnanya:
	x_1,1: 1
	x_1,2: 4
	x_1,3: 3
	x_1,4: 6
	x_2,1: 8
	x_2,2: 5
	x_2,3: 6
	x_2,4: 3
	y_1,1: 2
	y_1,2: 5
	y_1,3: 4
	y_1,4: 7
	y_2,1: 9
	y_2,2: 6
	y_2,3: 7
	y_2,4: 4
	
	Sisi dan Selisih Warnanya:
	x_1,1 - x_1,2: |1 - 4| = 3
	x_1,1 - x_1,4: |1 - 6| = 5
	x_1,1 - x_2,1: |1 - 8| = 7
	x_1,1 - y_1,1: |1 - 2| = 1
	x_1,2 - x_1,3: |4 - 3| = 1
	x_1,3 - x_1,4: |3 - 6| = 3
	x_2,1 - x_2,2: |8 - 5| = 3
	x_2,1 - x_2,4: |8 - 3| = 5
	x_2,1 - y_2,1: |8 - 9| = 1
	x_2,2 - x_2,3: |5 - 6| = 1
	x_2,3 - x_2,4: |6 - 3| = 3
	y_1,1 - y_1,2: |2 - 5| = 3
	y_1,1 - y_1,4: |2 - 7| = 5
	y_1,1 - y_2,1: |2 - 9| = 7
	y_1,2 - y_1,3: |5 - 4| = 1
	y_1,3 - y_1,4: |4 - 7| = 3
	y_2,1 - y_2,2: |9 - 6| = 3
	y_2,1 - y_2,4: |9 - 4| = 5
	y_2,2 - y_2,3: |6 - 7| = 1
	y_2,3 - y_2,4: |7 - 4| = 3

```
            self.E.add((f'y_{i},1', f'y_{i},{self.m}'))

        # Generate sisi penghubung antara x_(i,1)
dan y_(i,1)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan memenuhi
syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil dengan
memastikan bahwa simpul bertetangga
memiliki warna berbeda dan selisih ganjil.
        """
        self.color = {'x_1,1': 1, 'x_1,2': 4, 'x_1,4':
6, 'x_2,1': 8, 'x_2,2': 5, 'x_2,4': 3, 'y_1,1': 2,
'y_1,2': 5, 'y_1,4': 7, 'y_2,1': 9, 'y_2,2': 6,
'y_2,4': 4}  # Warna awal

        # Menentukan pewarnaan untuk x_1,3
        for c in colors:
          if (abs(c - self.color['x_1,2']) % 2 == 1
and
              abs(c - self.color['x_1,4']) % 2 == 1
and
              c != self.color['x_1,1'] and
              c != self.color['x_1,2'] and
              c != self.color['x_1,4']):
              self.color['x_1,3'] = c
              break

        # Menentukan pewarnaan untuk x_2,3
        for c in colors:
          if (abs(c - self.color['x_2,2']) % 2 == 1
and
```

```
            abs(c - self.color['x_2,4']) % 2 == 1
and
            c != self.color['x_2,1'] and
            c != self.color['x_2,2'] and
            c != self.color['x_2,4'] and
            abs(self.color['x_2,4'] - c) != 1):
            self.color['x_2,3'] = c
            break

        # Menentukan pewarnaan untuk y_1,3
        for c in colors:
          if (abs(c - self.color['y_1,2']) % 2 == 1
and
            abs(c - self.color['y_1,4']) % 2 == 1
and
            c != self.color['y_1,1'] and
            c != self.color['y_1,2'] and
            c != self.color['y_1,4']):
            self.color['y_1,3'] = c
            break

        # Menentukan pewarnaan untuk y_2,3
        for c in colors:
          if (abs(c - self.color['y_2,2']) % 2 == 1
and
            abs(c - self.color['y_2,4']) % 2 == 1
and
            c != self.color['y_2,1'] and
            c != self.color['y_2,2'] and
            c != self.color['y_2,4'] and
            abs(self.color['y_2,2'] - c) != 5 and
            abs(self.color['y_2,4'] - c) != 1):
            self.color['y_2,3'] = c
            break

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum
diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
```

```
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum diwarnai')

# Contoh eksekusi untuk n=2, m=4
n=2
m=4
colors = [1, 2, 3, 4, 5, 6, 7, 8, 9]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
    print("Tidak ditemukan pewarnaan yang
valid.")

graph.display()
```

1. j. Pewarnaan pada graf $L_3 \rhd C_m$, m merupakan bilangan genap dan $m \geq 4$

Kode

```
class Graph:
    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.V = set()
        self.E = set()
        self.color = {}

        # Generate graf
        self._generate_graph()

    def _generate_graph(self):
        """
        Membuat graf berdasarkan aturan yang
telah dijelaskan.
        """
        # Generate simpul x_(i,j) dan y_(i,j)
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                self.V.add(f'x_{i},{j}')
                self.V.add(f'y_{i},{j}')

        # Generate sisi horizontal antara x_(i,1)
dan x_(i+1,1), serta y_(i,1) dan y_(i+1,1)
        for i in range(1, self.n):
```

Output

```
Pewarnaan ditemukan!
Simpul dan Warnanya:
x_1,1: 3
x_1,2: 6
x_1,3: 7
x_1,4: 4
x_2,1: 10
x_2,2: 5
x_2,3: 2
x_2,4: 7
x_3,1: 9
x_3,2: 4
x_3,3: 1
x_3,4: 6
y_1,1: 8
y_1,2: 5
y_1,3: 4
y_1,4: 7
y_2,1: 1
y_2,2: 6
y_2,3: 9
y_2,4: 4
y_3,1: 2
y_3,2: 7
y_3,3: 10
y_3,4: 5
```

```
            self.E.add((f'x_{i},1', f'x_{i+1},1'))
            self.E.add((f'y_{i},1', f'y_{i+1},1'))

        # Generate sisi vertikal antara x_(i,j) dan
x_(i,j+1), serta y_(i,j) dan y_(i,j+1)
        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}',
f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}',
f'y_{i},{j+1}'))

        # Generate sisi langsung dari x_(i,1) ke
x_(i,m) dan y_(i,1) ke y_(i,m)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'x_{i},{self.m}'))
            self.E.add((f'y_{i},1', f'y_{i},{self.m}'))

        # Generate sisi penghubung antara x_(i,1)
dan y_(i,1)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan memenuhi
syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or self.color[u]
== self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil dengan
memastikan bahwa simpul bertetangga memiliki
warna berbeda dan selisih ganjil.
        """
        self.color = {'x_1,1': 3, 'x_1,2': 6, 'x_1,4': 4,
'x_2,1': 10, 'x_2,2': 5, 'x_2,4': 7, 'x_3,1': 9,
'x_3,2': 4, 'x_3,4': 6, 'y_1,1': 8, 'y_1,2': 5, 'y_1,4':
7, 'y_2,1': 1, 'y_2,2': 6, 'y_2,4': 4, 'y_3,1': 2,
'y_3,2': 7, 'y_3,4': 5}  # Warna awal
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |3 - 6| = 3
x_1,1 - x_1,4: |3 - 4| = 1
x_1,1 - x_2,1: |3 - 10| = 7
x_1,1 - y_1,1: |3 - 8| = 5
x_1,2 - x_1,3: |6 - 7| = 1
x_1,3 - x_1,4: |7 - 4| = 3
x_2,1 - x_2,2: |10 - 5| = 5
x_2,1 - x_2,4: |10 - 7| = 3
x_2,1 - x_3,1: |10 - 9| = 1
x_2,1 - y_2,1: |10 - 1| = 9
x_2,2 - x_2,3: |5 - 2| = 3
x_2,3 - x_2,4: |2 - 7| = 5
x_3,1 - x_3,2: |9 - 4| = 5
x_3,1 - x_3,4: |9 - 6| = 3
x_3,1 - y_3,1: |9 - 2| = 7
x_3,2 - x_3,3: |4 - 1| = 3
x_3,3 - x_3,4: |1 - 6| = 5
y_1,1 - y_1,2: |8 - 5| = 3
y_1,1 - y_1,4: |8 - 7| = 1
y_1,1 - y_2,1: |8 - 1| = 7
y_1,2 - y_1,3: |5 - 4| = 1
y_1,3 - y_1,4: |4 - 7| = 3
y_2,1 - y_2,2: |1 - 6| = 5
y_2,1 - y_2,4: |1 - 4| = 3
y_2,1 - y_3,1: |1 - 2| = 1
y_2,2 - y_2,3: |6 - 9| = 3
y_2,3 - y_2,4: |9 - 4| = 5
y_3,1 - y_3,2: |2 - 7| = 5
y_3,1 - y_3,4: |2 - 5| = 3
y_3,2 - y_3,3: |7 - 10| = 3
y_3,3 - y_3,4: |10 - 5| = 5
```

```
        # Menentukan pewarnaan untuk x_1,3
        for c in colors:
          if (abs(c - self.color['x_1,2']) % 2 == 1
and
              abs(c - self.color['x_1,4']) % 2 == 1 and
              c != self.color['x_1,1'] and
              c != self.color['x_1,2'] and
              c != self.color['x_1,4'] and
              abs(self.color['x_1,2'] - c) != 5 and
              abs(self.color['x_1,4'] - c) != 1):
              self.color['x_1,3'] = c
              break

        # Menentukan pewarnaan untuk x_2,3
        for c in colors:
          if (abs(c - self.color['x_2,2']) % 2 == 1
and
              abs(c - self.color['x_2,4']) % 2 == 1 and
              c != self.color['x_2,1'] and
              c != self.color['x_2,2'] and
              c != self.color['x_2,4']):
              self.color['x_2,3'] = c
              break

        # Menentukan pewarnaan untuk x_3,3
        for c in colors:
          if (abs(c - self.color['x_3,2']) % 2 == 1
and
              abs(c - self.color['x_3,4']) % 2 == 1 and
              c != self.color['x_3,1'] and
              c != self.color['x_3,2'] and
              c != self.color['x_3,4']):
              self.color['x_3,3'] = c
              break

        # Menentukan pewarnaan untuk y_1,3
        for c in colors:
          if (abs(c - self.color['y_1,2']) % 2 == 1
and
              abs(c - self.color['y_1,4']) % 2 == 1 and
              c != self.color['y_1,1'] and
              c != self.color['y_1,2'] and
              c != self.color['y_1,4'] and
              abs(self.color['y_1,2'] - c) != 3 and
              abs(self.color['y_1,4'] - c) != 1):
              self.color['y_1,3'] = c
              break
```

```
        # Menentukan pewarnaan untuk y_2,3
        for c in colors:
          if (abs(c - self.color['y_2,2']) % 2 == 1
and
              abs(c - self.color['y_2,4']) % 2 == 1 and
              c != self.color['y_2,1'] and
              c != self.color['y_2,2'] and
              c != self.color['y_2,4'] and
              abs(self.color['y_2,2'] - c) != 1 and
              abs(self.color['y_2,4'] - c) != 1):
              self.color['y_2,3'] = c
              break

        # Menentukan pewarnaan untuk y_3,3
        for c in colors:
          if (abs(c - self.color['y_3,2']) % 2 == 1
and
              abs(c - self.color['y_3,4']) % 2 == 1 and
              c != self.color['y_3,1'] and
              c != self.color['y_3,2'] and
              c != self.color['y_3,4'] and
              abs(self.color['y_3,2'] - c) != 1 and
              abs(self.color['y_3,4'] - c) != 1):
              self.color['y_3,3'] = c
              break

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum
diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum diwarnai')

# Contoh eksekusi untuk n=3, m=4
```

```
n=3
m=4
colors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
   print("Pewarnaan ditemukan!")
else:
   print("Tidak ditemukan pewarnaan yang
valid.")

graph.display()
```

1. k. Pewarnaan pada graf $L_4 \rhd C_m$, m merupakan bilangan genap dan $m \geq 4$

Kode

```
class Graph:
   def __init__(self, n, m):
      self.n = n
      self.m = m
      self.V = set()
      self.E = set()
      self.color = {}

      # Generate graf
      self._generate_graph()

   def _generate_graph(self):
      """
      Membuat graf berdasarkan aturan yang
telah dijelaskan.
      """
      # Generate simpul x_(i,j) dan y_(i,j)
      for i in range(1, self.n + 1):
         for j in range(1, self.m + 1):
            self.V.add(f'x_{i},{j}')
            self.V.add(f'y_{i},{j}')

      # Generate sisi horizontal antara
x_(i,1) dan x_(i+1,1), serta y_(i,1) dan
y_(i+1,1)
      for i in range(1, self.n):
         self.E.add((f'x_{i},1', f'x_{i+1},1'))
         self.E.add((f'y_{i},1', f'y_{i+1},1'))

      # Generate sisi vertikal antara x_(i,j)
dan x_(i,j+1), serta y_(i,j) dan y_(i,j+1)
      for i in range(1, self.n + 1):
         for j in range(1, self.m):
```

Output

```
Pewarnaan ditemukan!
Simpul dan Warnanya:
x_1,1: 3
x_1,2: 6
x_1,3: 7
x_1,4: 4
x_2,1: 10
x_2,2: 5
x_2,3: 8
x_2,4: 7
x_3,1: 11
x_3,2: 6
x_3,3: 9
x_3,4: 8
x_4,1: 4
x_4,2: 7
x_4,3: 8
x_4,4: 5
y_1,1: 8
y_1,2: 5
y_1,3: 4
y_1,4: 7
y_2,1: 1
y_2,2: 6
y_2,3: 3
y_2,4: 4
y_3,1: 2
y_3,2: 7
y_3,3: 4
y_3,4: 5
y_4,1: 9
y_4,2: 6
y_4,3: 7
y_4,4: 10
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |3 - 6| = 3
x_1,1 - x_1,4: |3 - 4| = 1
x_1,1 - x_2,1: |3 - 10| = 7
x_1,1 - y_1,1: |3 - 8| = 5
x_1,2 - x_1,3: |6 - 7| = 1
x_1,3 - x_1,4: |7 - 4| = 3
x_2,1 - x_2,2: |10 - 5| = 5
x_2,1 - x_2,4: |10 - 7| = 3
x_2,1 - x_3,1: |10 - 11| = 1
x_2,1 - y_2,1: |10 - 1| = 9
x_2,2 - x_2,3: |5 - 8| = 3
x_2,3 - x_2,4: |8 - 7| = 1
x_3,1 - x_3,2: |11 - 6| = 5
x_3,1 - x_3,4: |11 - 8| = 3
x_3,1 - x_4,1: |11 - 4| = 7
x_3,1 - y_3,1: |11 - 2| = 9
x_3,2 - x_3,3: |6 - 9| = 3
x_3,3 - x_3,4: |9 - 8| = 1
x_4,1 - x_4,2: |4 - 7| = 3
x_4,1 - x_4,4: |4 - 5| = 1
x_4,1 - y_4,1: |4 - 9| = 5
x_4,2 - x_4,3: |7 - 8| = 1
x_4,3 - x_4,4: |8 - 5| = 3
```

```
            self.E.add((f'x_{i},{j}',
f'x_{i},{j+1}'))
            self.E.add((f'y_{i},{j}',
f'y_{i},{j+1}'))

        # Generate sisi langsung dari x_(i,1) ke
x_(i,m) dan y_(i,1) ke y_(i,m)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1',
f'x_{i},{self.m}'))
            self.E.add((f'y_{i},1',
f'y_{i},{self.m}'))

        # Generate sisi penghubung antara
x_(i,1) dan y_(i,1)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda dan
selisih ganjil.
        """
        self.color = {'x_1,1': 3, 'x_1,2': 6,
'x_1,4': 4, 'x_2,1': 10, 'x_2,2': 5, 'x_2,4': 7,
'x_3,1': 11, 'x_3,2': 6, 'x_3,4': 8, 'x_4,1': 4,
'x_4,2': 7, 'x_4,4': 5, 'y_1,1': 8, 'y_1,2': 5,
'y_1,4': 7, 'y_2,1': 1, 'y_2,2': 6, 'y_2,4': 4,
'y_3,1': 2, 'y_3,2': 7, 'y_3,4': 5, 'y_4,1': 9,
'y_4,2': 6, 'y_4,4': 10}

        # Menentukan pewarnaan untuk x_1,3
```

```
y_1,1 - y_1,2: |8 - 5| = 3
y_1,1 - y_1,4: |8 - 7| = 1
y_1,1 - y_2,1: |8 - 1| = 7
y_1,2 - y_1,3: |5 - 4| = 1
y_1,3 - y_1,4: |4 - 7| = 3
y_2,1 - y_2,2: |1 - 6| = 5
y_2,1 - y_2,4: |1 - 4| = 3
y_2,1 - y_3,1: |1 - 2| = 1
y_2,2 - y_2,3: |6 - 3| = 3
y_2,3 - y_2,4: |3 - 4| = 1
y_3,1 - y_3,2: |2 - 7| = 5
y_3,1 - y_3,4: |2 - 5| = 3
y_3,1 - y_4,1: |2 - 9| = 7
y_3,2 - y_3,3: |7 - 4| = 3
y_3,3 - y_3,4: |4 - 5| = 1
y_4,1 - y_4,2: |9 - 6| = 3
y_4,1 - y_4,4: |9 - 10| = 1
y_4,2 - y_4,3: |6 - 7| = 1
y_4,3 - y_4,4: |7 - 10| = 3
```

```
        for c in colors:
          if (abs(c - self.color['x_1,2']) % 2 ==
1 and
              abs(c - self.color['x_1,4']) % 2 == 1
and
              c != self.color['x_1,1'] and
              c != self.color['x_1,2'] and
              c != self.color['x_1,4'] and
              c != self.color['x_1,1'] and
              abs(self.color['x_1,2'] - c) != 5 and
              abs(self.color['x_1,4'] - c) != 1):
              self.color['x_1,3'] = c
              break

        # Menentukan pewarnaan untuk x_2,3
        for c in colors:
          if (abs(c - self.color['x_2,2']) % 2 ==
1 and
              abs(c - self.color['x_2,4']) % 2 == 1
and
              c != self.color['x_2,1'] and
              c != self.color['x_2,1'] and
              c != self.color['x_2,4'] and
              c != self.color['x_2,1'] and
              abs(self.color['x_2,4'] - c) != 5 and
              abs(self.color['x_2,2'] - c) != 1):
              self.color['x_2,3'] = c
              break

        # Menentukan pewarnaan untuk x_3,3
        for c in colors:
          if (abs(c - self.color['x_3,2']) % 2 ==
1 and
              abs(c - self.color['x_3,4']) % 2 == 1
and
              c != self.color['x_3,1'] and
              c != self.color['x_3,2'] and
              c != self.color['x_3,4'] and
              c != self.color['x_3,1'] and
              abs(self.color['x_3,2'] - c) != 5 and
              abs(self.color['x_3,2'] - c) != 1 and
              abs(self.color['x_3,4'] - c) != 7 and
              abs(self.color['x_3,4'] - c) != 5):
              self.color['x_3,3'] = c
              break

        # Menentukan pewarnaan untuk x_4,3
        for c in colors:
```

```
        if (abs(c - self.color['x_4,2']) % 2 ==
1 and
            abs(c - self.color['x_4,4']) % 2 == 1
and
            c != self.color['x_4,1'] and
            c != self.color['x_4,2'] and
            c != self.color['x_4,4'] and
            c != self.color['x_4,1'] and
            abs(self.color['x_4,2'] - c) != 5 and
            abs(self.color['x_4,4'] - c) != 1):
            self.color['x_4,3'] = c
            break

      # Menentukan pewarnaan untuk y_1,3
      for c in colors:
        if (abs(c - self.color['y_1,2']) % 2 ==
1 and
            abs(c - self.color['y_1,4']) % 2 == 1
and
            c != self.color['y_1,1'] and
            c != self.color['y_1,2'] and
            c != self.color['y_1,4'] and
            c != self.color['y_1,1'] and
            abs(self.color['y_1,2'] - c) != 3 and
            abs(self.color['y_1,4'] - c) != 5):
            self.color['y_1,3'] = c
            break

      # Menentukan pewarnaan untuk y_2,3
      for c in colors:
        if (abs(c - self.color['y_2,2']) % 2 ==
1 and
            abs(c - self.color['y_2,4']) % 2 == 1
and
            c != self.color['y_2,1'] and
            c != self.color['y_2,2'] and
            c != self.color['y_2,4'] and
            c != self.color['y_2,1']):
            self.color['y_2,3'] = c
            break

      # Menentukan pewarnaan untuk y_3,3
      for c in colors:
        if (abs(c - self.color['y_3,2']) % 2 ==
1 and
            abs(c - self.color['y_3,4']) % 2 == 1
and
            c != self.color['y_3,1'] and
```

```
            c != self.color['y_3,2'] and
            c != self.color['y_3,4'] and
            c != self.color['y_3,1']):
            self.color['y_3,3'] = c
            break

        # Menentukan pewarnaan untuk y_4,3
        for c in colors:
          if (abs(c - self.color['y_4,2']) % 2 ==
1 and
            abs(c - self.color['y_4,4']) % 2 == 1
and
            c != self.color['y_4,1'] and
            c != self.color['y_4,2'] and
            c != self.color['y_4,4'] and
            c != self.color['y_4,1'] and
            abs(self.color['y_4,2'] - c) != 3 and
            abs(self.color['y_4,4'] - c) != 9 and
            abs(self.color['y_4,4'] - c) != 7 and
            abs(self.color['y_4,4'] - c) != 5):
            self.color['y_4,3'] = c
            break

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum
diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')

# Contoh eksekusi untuk n=4, m=4
n=4
m=4
```

```
colors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
    print("Tidak ditemukan pewarnaan yang valid.")

graph.display()
```

1. l. Pewarnaan pada graf $L_5 \rhd C_m$, m merupakan bilangan genap dan $m \geq 4$

Kode

```
class Graph:
    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.V = set()
        self.E = set()
        self.color = {}

        # Generate graf
        self._generate_graph()

    def _generate_graph(self):
        """
        Membuat graf berdasarkan aturan yang telah dijelaskan.
        """
        # Generate simpul x_(i,j) dan y_(i,j)
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                self.V.add(f'x_{i},{j}')
                self.V.add(f'y_{i},{j}')

        # Generate sisi horizontal antara x_(i,1) dan x_(i+1,1), serta y_(i,1) dan y_(i+1,1)
        for i in range(1, self.n):
            self.E.add((f'x_{i},1', f'x_{i+1},1'))
            self.E.add((f'y_{i},1', f'y_{i+1},1'))

        # Generate sisi vertikal antara x_(i,j) dan x_(i,j+1), serta y_(i,j) dan y_(i,j+1)
        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}', f'x_{i},{j+1}'))
```

Output

```
Pewarnaan ditemukan!
Simpul dan Warnanya:
x_1,1: 8
x_1,2: 1
x_1,3: 2
x_1,4: 9
x_2,1: 3
x_2,2: 6
x_2,3: 5
x_2,4: 2
x_3,1: 12
x_3,2: 7
x_3,3: 10
x_3,4: 9
x_4,1: 11
x_4,2: 6
x_4,3: 9
x_4,4: 8
x_5,1: 4
x_5,2: 7
x_5,3: 8
x_5,4: 5
y_1,1: 5
y_1,2: 12
y_1,3: 11
y_1,4: 4
y_2,1: 10
y_2,2: 7
y_2,3: 6
y_2,4: 9
y_3,1: 1
y_3,2: 6
y_3,3: 3
y_3,4: 4
y_4,1: 2
y_4,2: 7
y_4,3: 4
y_4,4: 5
y_5,1: 9
y_5,2: 6
y_5,3: 5
y_5,4: 8
```

```
            self.E.add((f'y_{i},{j}',
f'y_{i},{j+1}'))

        # Generate sisi langsung dari x_(i,1) ke
x_(i,m) dan y_(i,1) ke y_(i,m)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1',
f'x_{i},{self.m}'))
            self.E.add((f'y_{i},1',
f'y_{i},{self.m}'))

        # Generate sisi penghubung antara
x_(i,1) dan y_(i,1)
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda dan
selisih ganjil.
        """
        self.color = {'x_1,1': 8, 'x_1,2': 1,
'x_1,4': 9, 'x_2,1': 3, 'x_2,2': 6, 'x_2,4': 2,
'x_3,1': 12, 'x_3,2': 7, 'x_3,4': 9, 'x_4,1': 11,
'x_4,2': 6, 'x_4,4': 8, 'x_5,1': 4, 'x_5,2': 7,
'x_5,4': 5, 'y_1,1': 5, 'y_1,2': 12, 'y_1,4': 4,
'y_2,1': 10,  'y_2,2': 7, 'y_2,4': 9, 'y_3,1': 1,
'y_3,2': 6, 'y_3,4': 4, 'y_4,1': 2, 'y_4,2': 7,
'y_4,4': 5, 'y_5,1': 9, 'y_5,2': 6, 'y_5,4': 8}

        # Menentukan pewarnaan untuk x_1,3
        for c in colors:
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |8 - 1| = 7
x_1,1 - x_1,4: |8 - 9| = 1
x_1,1 - x_2,1: |8 - 3| = 5
x_1,1 - y_1,1: |8 - 5| = 3
x_1,2 - x_1,3: |1 - 2| = 1
x_1,3 - x_1,4: |2 - 9| = 7
x_2,1 - x_2,2: |3 - 6| = 3
x_2,1 - x_2,4: |3 - 2| = 1
x_2,1 - x_3,1: |3 - 12| = 9
x_2,1 - y_2,1: |3 - 10| = 7
x_2,2 - x_2,3: |6 - 5| = 1
x_2,3 - x_2,4: |5 - 2| = 3
x_3,1 - x_3,2: |12 - 7| = 5
x_3,1 - x_3,4: |12 - 9| = 3
x_3,1 - x_4,1: |12 - 11| = 1
x_3,1 - y_3,1: |12 - 1| = 11
x_3,2 - x_3,3: |7 - 10| = 3
x_3,3 - x_3,4: |10 - 9| = 1
x_4,1 - x_4,2: |11 - 6| = 5
x_4,1 - x_4,4: |11 - 8| = 3
x_4,1 - x_5,1: |11 - 4| = 7
x_4,1 - y_4,1: |11 - 2| = 9
x_4,2 - x_4,3: |6 - 9| = 3
x_4,3 - x_4,4: |9 - 8| = 1
x_5,1 - x_5,2: |4 - 7| = 3
x_5,1 - x_5,4: |4 - 5| = 1
x_5,1 - y_5,1: |4 - 9| = 5
x_5,2 - x_5,3: |7 - 8| = 1
x_5,3 - x_5,4: |8 - 5| = 3
y_1,1 - y_1,2: |5 - 12| = 7
y_1,1 - y_1,4: |5 - 4| = 1
y_1,1 - y_2,1: |5 - 10| = 5
y_1,2 - y_1,3: |12 - 11| = 1
y_1,3 - y_1,4: |11 - 4| = 7
y_2,1 - y_2,2: |10 - 7| = 3
y_2,1 - y_2,4: |10 - 9| = 1
y_2,1 - y_3,1: |10 - 1| = 9
y_2,2 - y_2,3: |7 - 6| = 1
y_2,3 - y_2,4: |6 - 9| = 3
y_3,1 - y_3,2: |1 - 6| = 5
y_3,1 - y_3,4: |1 - 4| = 3
y_3,1 - y_4,1: |1 - 2| = 1
y_3,2 - y_3,3: |6 - 3| = 3
y_3,3 - y_3,4: |3 - 4| = 1
y_4,1 - y_4,2: |2 - 7| = 5
y_4,1 - y_4,4: |2 - 5| = 3
y_4,1 - y_5,1: |2 - 9| = 7
y_4,2 - y_4,3: |7 - 4| = 3
y_4,3 - y_4,4: |4 - 5| = 1
y_5,1 - y_5,2: |9 - 6| = 3
y_5,1 - y_5,4: |9 - 8| = 1
y_5,2 - y_5,3: |6 - 5| = 1
y_5,3 - y_5,4: |5 - 8| = 3
```

```
        if (abs(c - self.color['x_1,2']) % 2 == 1
and
            abs(c - self.color['x_1,4']) % 2 == 1
and
            c != self.color['x_1,2'] and
            c != self.color['x_1,4'] and
            c != self.color['x_1,1'] and
            abs(self.color['x_1,4'] - c) != 1 and
            abs(self.color['x_1,4'] - c) != 3 and
            abs(self.color['x_1,4'] - c) != 5):
            self.color['x_1,3'] = c
            break

      # Menentukan pewarnaan untuk x_2,3
      for c in colors:
        if (abs(c - self.color['x_2,2']) % 2 == 1
and
            abs(c - self.color['x_2,4']) % 2 == 1
and
            c != self.color['x_2,2'] and
            c != self.color['x_2,4'] and
            c != self.color['x_2,1'] and
            abs(self.color['x_2,2'] - c) != 3 and
            abs(self.color['x_2,4'] - c) != 1):
            self.color['x_2,3'] = c
            break

      # Menentukan pewarnaan untuk x_3,3
      for c in colors:
        if (abs(c - self.color['x_3,2']) % 2 == 1
and
            abs(c - self.color['x_3,4']) % 2 == 1
and
            c != self.color['x_3,2'] and
            c != self.color['x_3,4'] and
            c != self.color['x_3,1'] and
            abs(self.color['x_3,2'] - c) != 1 and
            abs(self.color['x_3,2'] - c) != 5 and
            abs(self.color['x_3,4'] - c) != 3 and
            abs(self.color['x_3,4'] - c) != 5):
            self.color['x_3,3'] = c
            break

      # Menentukan pewarnaan untuk x_4,3
      for c in colors:
        if (abs(c - self.color['x_4,2']) % 2 == 1
and
```

```
            abs(c - self.color['x_4,4']) % 2 == 1
and
            c != self.color['x_4,2'] and
            c != self.color['x_4,4'] and
            c != self.color['x_4,1'] and
            abs(self.color['x_4,2'] - c) != 1 and
            abs(self.color['x_4,2'] - c) != 5 and
            abs(self.color['x_4,4'] - c) != 3 and
            abs(self.color['x_4,4'] - c) != 5):
            self.color['x_4,3'] = c
            break

        # Menentukan pewarnaan untuk x_5,3
        for c in colors:
          if (abs(c - self.color['x_5,2']) % 2 == 1
and
            abs(c - self.color['x_5,4']) % 2 == 1
and
            c != self.color['x_5,2'] and
            c != self.color['x_5,4'] and
            c != self.color['x_5,1'] and
            abs(self.color['x_5,2'] - c) != 5 and
            abs(self.color['x_5,4'] - c) != 1):
            self.color['x_5,3'] = c
            break

        # Menentukan pewarnaan untuk y_1,3
        for c in colors:
          if (abs(c - self.color['y_1,2']) % 2 == 1
and
            abs(c - self.color['y_1,4']) % 2 == 1
and
            c != self.color['y_1,2'] and
            c != self.color['y_1,4'] and
            c != self.color['y_1,1'] and
            abs(self.color['y_1,4'] - c) != 1 and
            abs(self.color['y_1,4'] - c) != 3 and
            abs(self.color['y_1,4'] - c) != 5):
            self.color['y_1,3'] = c
            break

        # Menentukan pewarnaan untuk y_2,3
        for c in colors:
          if (abs(c - self.color['y_2,2']) % 2 == 1
and
            abs(c - self.color['y_2,4']) % 2 == 1
and
            c != self.color['y_2,2'] and
```

```
            c != self.color['y_2,4'] and
            c != self.color['y_2,1'] and
            abs(self.color['y_2,2'] - c) != 3 and
            abs(self.color['y_2,4'] - c) != 5 and
            abs(self.color['y_2,4'] - c) != 7):
            self.color['y_2,3'] = c
            break

        # Menentukan pewarnaan untuk y_3,2
        for c in colors:
          if (abs(c - self.color['y_3,2']) % 2 == 1
and
            abs(c - self.color['y_3,4']) % 2 == 1
and
            c != self.color['y_3,2'] and
            c != self.color['y_3,4'] and
            c != self.color['y_3,1'] and
            abs(self.color['y_3,2'] - c) != 1 and
            abs(self.color['y_3,4'] - c) != 3):
            self.color['y_3,3'] = c
            break

        # Menentukan pewarnaan untuk y_4,3
        for c in colors:
          if (abs(c - self.color['y_4,2']) % 2 == 1
and
            abs(c - self.color['y_4,4']) % 2 == 1
and
            c != self.color['y_4,2'] and
            c != self.color['y_4,4'] and
            c != self.color['y_4,1'] and
            abs(self.color['y_4,2'] - c) != 1 and
            abs(self.color['y_4,4'] - c) != 3):
            self.color['y_4,3'] = c
            break

        # Menentukan pewarnaan untuk y_5,3
        for c in colors:
          if (abs(c - self.color['y_5,2']) % 2 == 1
and
            abs(c - self.color['y_5,4']) % 2 == 1
and
            c != self.color['y_5,2'] and
            c != self.color['y_5,4'] and
            c != self.color['y_5,1'] and
            abs(self.color['y_5,2'] - c) != 3 and
            abs(self.color['y_5,2'] - c) != 5 and
            abs(self.color['y_5,4'] - c) != 1):
```

self.color['y_5,3'] = c
break
 	
return self.is_valid_coloring()
 	
def display(self):
"""
Menampilkan simpul, sisi, dan warnanya.
"""
print("Simpul dan Warnanya:")
for v in sorted(self.V):
print(f'{v}: {self.color.get(v, "Belum diwarnai")}')
 	
print("\nSisi dan Selisih Warnanya:")
for u, v in sorted(self.E):
if u in self.color and v in self.color:
print(f'{u} - {v}: |{self.color[u]} - {self.color[v]}| = {abs(self.color[u] - self.color[v])}')
else:
print(f'{u} - {v}: Belum diwarnai')
 	
# Contoh eksekusi untuk n=5, m=4 	
n=5
m=4
colors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
print("Pewarnaan ditemukan!")
else:
print("Tidak ditemukan pewarnaan yang valid.")
 	
graph.display()	

1. m. Pewarnaan pada graf $L_n \triangleright C_m$, m merupakan bilangan genap, $n \geq 6$, dan $m \geq 4$

Kode	Output

```
class Graph:
    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.V = set()
        self.E = set()
        self.color = {}

        self._generate_graph()

    def _generate_graph(self):
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                self.V.add(f'x_{i},{j}')
                self.V.add(f'y_{i},{j}')

        for i in range(1, self.n):
            self.E.add((f'x_{i},1', f'x_{i+1},1'))
            self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},{j}',
f'x_{i},{j+1}'))
                self.E.add((f'y_{i},{j}',
f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1',
f'x_{i},{self.m}'))
            self.E.add((f'y_{i},1',
f'y_{i},{self.m}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def apply_coloring_rule(self):
        """
        Menerapkan aturan pewarnaan sesuai
dengan rumus yang diberikan untuk simpul
v.
        """
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                # Pewarnaan untuk x_(i,j)
                if i % 4 == 0 and j % 4 == 1:
                    self.color[f'x_{i},{j}'] = 1
                elif i % 4 == 0 and j % 4 == 2:
```

```
Simpul dan Warnanya:
x_1,1: 10
x_1,2: 7
x_1,3: 8
x_1,4: 5
x_2,1: 3
x_2,2: 6
x_2,3: 5
x_2,4: 8
x_3,1: 12
x_3,2: 9
x_3,3: 10
x_3,4: 7
x_4,1: 1
x_4,2: 4
x_4,3: 3
x_4,4: 6
x_5,1: 10
x_5,2: 7
x_5,3: 8
x_5,4: 5
x_6,1: 3
x_6,2: 6
x_6,3: 5
x_6,4: 8
y_1,1: 11
y_1,2: 8
y_1,3: 9
y_1,4: 6
y_2,1: 4
y_2,2: 7
y_2,3: 6
y_2,4: 9
y_3,1: 13
y_3,2: 10
y_3,3: 11
y_3,4: 8
y_4,1: 2
y_4,2: 5
y_4,3: 4
y_4,4: 7
y_5,1: 11
y_5,2: 8
y_5,3: 9
y_5,4: 6
y_6,1: 4
y_6,2: 7
y_6,3: 6
y_6,4: 9
```

```
            self.color[f'x_{i},{j}'] = 4
        elif i % 4 == 0 and j % 4 == 3:
            self.color[f'x_{i},{j}'] = 3
        elif i % 4 == 0 and j % 4 == 0:
            self.color[f'x_{i},{j}'] = 6
        elif i % 4 == 1 and j % 4 == 0:
            self.color[f'x_{i},{j}'] = 5
        elif i % 4 == 1 and j % 4 == 1:
            self.color[f'x_{i},{j}'] = 10
        elif i % 4 == 1 and j % 4 == 2:
            self.color[f'x_{i},{j}'] = 7
        elif i % 4 == 1 and j % 4 == 3:
            self.color[f'x_{i},{j}'] = 8
        elif i % 4 == 2 and j % 4 == 0:
            self.color[f'x_{i},{j}'] = 8
        elif i % 4 == 2 and j % 4 == 1:
            self.color[f'x_{i},{j}'] = 3
        elif i % 4 == 2 and j % 4 == 2:
            self.color[f'x_{i},{j}'] = 6
        elif i % 4 == 2 and j % 4 == 3:
            self.color[f'x_{i},{j}'] = 5
        elif i % 4 == 3 and j % 4 == 0:
            self.color[f'x_{i},{j}'] = 7
        elif i % 4 == 3 and j % 4 == 1:
            self.color[f'x_{i},{j}'] = 12
        elif i % 4 == 3 and j % 4 == 2:
            self.color[f'x_{i},{j}'] = 9
        elif i % 4 == 3 and j % 4 == 3:
            self.color[f'x_{i},{j}'] = 10

        # Pewarnaan untuk y_(i,j)
        if i % 4 == 0 and j % 4 == 0:
            self.color[f'y_{i},{j}'] = 7
        elif i % 4 == 0 and j % 4 == 1:
            self.color[f'y_{i},{j}'] = 2
        elif i % 4 == 0 and j % 4 == 2:
            self.color[f'y_{i},{j}'] = 5
        elif i % 4 == 0 and j % 4 == 3:
            self.color[f'y_{i},{j}'] = 4
        elif i % 4 == 1 and j % 4 == 0:
            self.color[f'y_{i},{j}'] = 6
        elif i % 4 == 1 and j % 4 == 1:
            self.color[f'y_{i},{j}'] = 11
        elif i % 4 == 1 and j % 4 == 2:
            self.color[f'y_{i},{j}'] = 8
        elif i % 4 == 1 and j % 4 == 3:
            self.color[f'y_{i},{j}'] = 9
        elif i % 4 == 2 and j % 4 == 0:
```

```
x_1,1 - x_1,2: |10 - 7| = 3
x_1,1 - x_1,4: |10 - 5| = 5
x_1,1 - x_2,1: |10 - 3| = 7
x_1,1 - y_1,1: |10 - 11| = 1
x_1,2 - x_1,3: |7 - 8| = 1
x_1,3 - x_1,4: |8 - 5| = 3
x_2,1 - x_2,2: |3 - 6| = 3
x_2,1 - x_2,4: |3 - 8| = 5
x_2,1 - x_3,1: |3 - 12| = 9
x_2,1 - y_2,1: |3 - 4| = 1
x_2,2 - x_2,3: |6 - 5| = 1
x_2,3 - x_2,4: |5 - 8| = 3
x_3,1 - x_3,2: |12 - 9| = 3
x_3,1 - x_3,4: |12 - 7| = 5
x_3,1 - x_4,1: |12 - 1| = 11
x_3,1 - y_3,1: |12 - 13| = 1
x_3,2 - x_3,3: |9 - 10| = 1
x_3,3 - x_3,4: |10 - 7| = 3
x_4,1 - x_4,2: |1 - 4| = 3
x_4,1 - x_4,4: |1 - 6| = 5
x_4,1 - x_5,1: |1 - 10| = 9
x_4,1 - y_4,1: |1 - 2| = 1
x_4,2 - x_4,3: |4 - 3| = 1
x_4,3 - x_4,4: |3 - 6| = 3
x_5,1 - x_5,2: |10 - 7| = 3
x_5,1 - x_5,4: |10 - 5| = 5
x_5,1 - x_6,1: |10 - 3| = 7
x_5,1 - y_5,1: |10 - 11| = 1
x_5,2 - x_5,3: |7 - 8| = 1
x_5,3 - x_5,4: |8 - 5| = 3
x_6,1 - x_6,2: |3 - 6| = 3
x_6,1 - x_6,4: |3 - 8| = 5
x_6,1 - y_6,1: |3 - 4| = 1
x_6,2 - x_6,3: |6 - 5| = 1
x_6,3 - x_6,4: |5 - 8| = 3
y_1,1 - y_1,2: |11 - 8| = 3
y_1,1 - y_1,4: |11 - 6| = 5
y_1,1 - y_2,1: |11 - 4| = 7
y_1,2 - y_1,3: |8 - 9| = 1
y_1,3 - y_1,4: |9 - 6| = 3
y_2,1 - y_2,2: |4 - 7| = 3
y_2,1 - y_2,4: |4 - 9| = 5
y_2,1 - y_3,1: |4 - 13| = 9
y_2,2 - y_2,3: |7 - 6| = 1
y_2,3 - y_2,4: |6 - 9| = 3
y_3,1 - y_3,2: |13 - 10| = 3
y_3,1 - y_3,4: |13 - 8| = 5
y_3,1 - y_4,1: |13 - 2| = 11
y_3,2 - y_3,3: |10 - 11| = 1
y_3,3 - y_3,4: |11 - 8| = 3
y_4,1 - y_4,2: |2 - 5| = 3
y_4,1 - y_4,4: |2 - 7| = 5
y_4,1 - y_5,1: |2 - 11| = 9
y_4,2 - y_4,3: |5 - 4| = 1
y_4,3 - y_4,4: |4 - 7| = 3
y_5,1 - y_5,2: |11 - 8| = 3
y_5,1 - y_5,4: |11 - 6| = 5
y_5,1 - y_6,1: |11 - 4| = 7
y_5,2 - y_5,3: |8 - 9| = 1
y_5,3 - y_5,4: |9 - 6| = 3
y_6,1 - y_6,2: |4 - 7| = 3
y_6,1 - y_6,4: |4 - 9| = 5
y_6,2 - y_6,3: |7 - 6| = 1
y_6,3 - y_6,4: |6 - 9| = 3
```

```
                self.color[f'y_{i},{j}'] = 9
            elif i % 4 == 2 and j % 4 == 1:
                self.color[f'y_{i},{j}'] = 4
            elif i % 4 == 2 and j % 4 == 2:
                self.color[f'y_{i},{j}'] = 7
            elif i % 4 == 2 and j % 4 == 3:
                self.color[f'y_{i},{j}'] = 6
            elif i % 4 == 3 and j % 4 == 0:
                self.color[f'y_{i},{j}'] = 8
            elif i % 4 == 3 and j % 4 == 1:
                self.color[f'y_{i},{j}'] = 13
            elif i % 4 == 3 and j % 4 == 2:
                self.color[f'y_{i},{j}'] = 10
            elif i % 4 == 3 and j % 4 == 3:
                self.color[f'y_{i},{j}'] = 11

    def display(self):
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum
diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')

# Contoh eksekusi untuk n = 6, m = ...
n = 6
m = 4
graph = Graph(n, m)
graph.apply_coloring_rule()
graph.display()
```

1. n. Pewarnaan pada graf $L_1 \rhd S_m, m \geq 2$

Langkah-langkah yang digunakan, yaitu:
1. Membuat simpul dan sisi.
2. Memastikan bahwa titik yang bertetangga tidak memiliki warna yang sama serta sisi yang terinduksi adalah ganjil.
3. Menambahkan aturan manual agar dua sisi yang bertetangga tidak sama.
4. Menampilkan hasil yang didapatkan dengan beberapa kali percobaan.
5. Menampilkan gambar dari hasil yang didapatkan.

Kode	Output

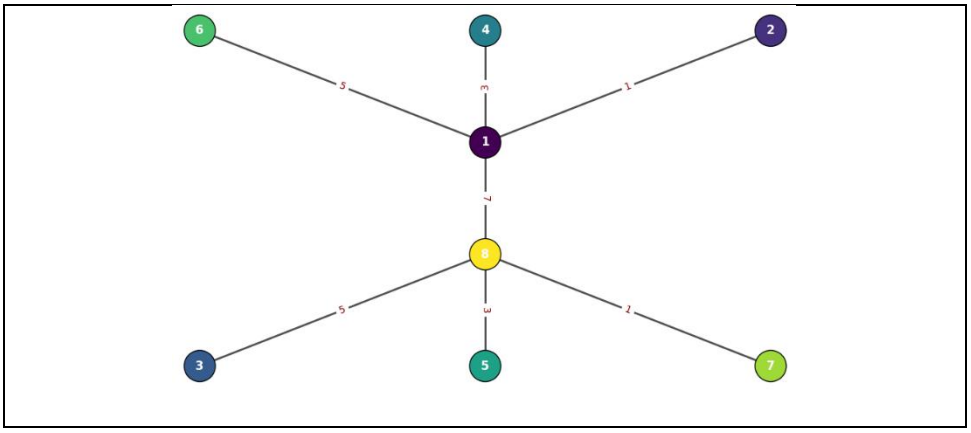
from itertools import permutations, product
import networkx as nx
import matplotlib.pyplot as plt
import math
 	
class GraphColoring:
  def __init__(self, n, m, colors):
  self.n = n
  self.m = m
  self.colors = colors
  self.nodes = [f'x_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)] + \
  [f'y_{i},{j}' for i in range(1, n+1) for j in range(1, m+1)]
  self.E = set()
  self.custom_edge_constraints = []
  self.color = {}
 	
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
 	
  for i in range(1, self.n + 1):
  for j in range(1, self.m):
  self.E.add((f'x_{i},1', f'x_{i},{j+1}'))
  self.E.add((f'y_{i},1', f'y_{i},{j+1}'))
 	
  for i in range(1, self.n + 1):
  self.E.add((f'x_{i},1', f'y_{i},1'))
 	
  def is_valid_coloring(self, coloring):
  for u, v in self.E:
  if u in coloring and v in coloring:
  if coloring[u] == coloring[v]:
  return False
  if abs(coloring[u] - coloring[v]) % 2 == 0:
  return False
  return True
 	
  def check_custom_constraints(self, coloring):
  for (u1, v1), (u2, v2) in self.custom_edge_constraints:
	Solusi ditemukan:
	x_1,1 = 1
	x_1,2 = 2
	x_1,3 = 4
	x_1,4 = 6
	y_1,1 = 8
	y_1,2 = 7
	y_1,3 = 5
	y_1,4 = 3
	
	Selisih warna sisi:
	x_1,1 - x_1,2 : |1 - 2| =
	x_1,1 - x_1,3 : |1 - 4| =
	x_1,1 - x_1,4 : |1 - 6| =
	x_1,1 - y_1,1 : |1 - 8| =
	y_1,1 - y_1,2 : |8 - 7| =
	y_1,1 - y_1,3 : |8 - 5| =
	y_1,1 - y_1,4 : |8 - 3| =

```
            if all(k in coloring for k in [u1, v1,
u2, v2]):
                d1 = abs(coloring[u1] -
coloring[v1])
                d2 = abs(coloring[u2] -
coloring[v2])
                if d1 == d2:
                    return False
        return True

    def print_solution(self, coloring):
        self.color = coloring
        print("Solusi ditemukan:")
        for node in sorted(self.nodes):
            print(f"{node} = {coloring[node]}")
        print("\nSelisih warna sisi:")
        for u, v in sorted(self.E):
            print(f"{u} - {v} : |{coloring[u]} -
{coloring[v]}| = {abs(coloring[u] -
coloring[v])}")

        self.draw()

    def solve(self, max_solutions=1):
        valid_solutions = []
        total_checked = 0

        for coloring in product(self.colors,
repeat=len(self.nodes)):
            total_checked += 1
            coloring_dict = dict(zip(self.nodes,
coloring))
            if
self.is_valid_coloring(coloring_dict) and
self.check_custom_constraints(coloring_dict
):
                valid_solutions.append(coloring_d
ict)
                print(f"\nSolusi ke-
{len(valid_solutions)}:")
                self.color = coloring_dict
                self.print_solution(coloring_dict)
                if len(valid_solutions) >=
max_solutions:
                    break

        if not valid_solutions:
```

```
            print(f"Tidak ada pewarnaan valid
dari {total_checked} kemungkinan yang
diperiksa.")

    def add_constraint(self, edge1, edge2):
        self.custom_edge_constraints.append((
edge1, edge2))

    def draw(self):
        G = nx.Graph()
        G.add_nodes_from(self.nodes)
        G.add_edges_from(self.E)

        # Posisi simpul: x_(i,j) di atas, y_(i,j)
di bawah
        pos = {
            'x_1,4': (-2, 2),
            'x_1,3': (0, 2),
            'x_1,2': (2, 2),
            'x_1,1': (0, 1),
            'y_1,1': (0, 0),
            'y_1,2': (2, -1),
            'y_1,3': (0, -1),
            'y_1,4': (-2, -1),
        }

        # Label simpul: nilai warna
        node_labels = {v: str(self.color.get(v,
'?')) for v in G.nodes}

        # Warna simpul berdasarkan
pewarnaan
        values = [self.color.get(v, 0.25) for v in
G.nodes]
        cmap = plt.cm.viridis

        plt.figure(figsize=(10, 6))
        nx.draw_networkx_nodes(G, pos,
node_color=values, cmap=cmap,
node_size=1000, edgecolors='black')
        nx.draw_networkx_labels(G, pos,
labels=node_labels, font_color='white',
font_weight='bold')
        nx.draw_networkx_edges(G, pos,
width=2, alpha=0.7)

        # Label sisi = selisih warna
        edge_labels = {}
```

```
        for u, v in G.edges:
            if u in self.color and v in self.color:
                edge_labels[(u, v)] =
str(abs(self.color[u] - self.color[v]))
        nx.draw_networkx_edge_labels(G,
pos, edge_labels=edge_labels,
font_color='darkred', font_size=10)

        plt.title('')
        plt.axis('off')
        plt.tight_layout()
        plt.show()

# Buat objek untuk n = 1
gc = GraphColoring(n=1, m=4, colors=[1, 2,
3, 4, 5, 6, 7, 8])

gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1',
'x_1,2'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1',
'x_1,3'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('x_1,1',
'x_1,4'))

gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1',
'x_1,3'))
gc.add_constraint(('x_1,1', 'x_1,2'), ('x_1,1',
'x_1,4'))
gc.add_constraint(('x_1,1', 'x_1,3'), ('x_1,1',
'x_1,4'))

gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1',
'y_1,2'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1',
'y_1,3'))
gc.add_constraint(('x_1,1', 'y_1,1'), ('y_1,1',
'y_1,4'))

gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1',
'y_1,3'))
gc.add_constraint(('y_1,1', 'y_1,2'), ('y_1,1',
'y_1,4'))
gc.add_constraint(('y_1,1', 'y_1,3'), ('y_1,1',
'y_1,4'))

# Jalankan
gc.solve(max_solutions=6)
```

Output Gambar



1. o. Pewarnaan pada graf $L_2 \rhd S_m$, $m \geq 2$

Kode	Output

class Graph:
  def __init__(self, n, m):
  self.n = n
  self.m = m
  self.V = set()
  self.E = set()
  self.color = {}
 	
  # Generate graf
  self._generate_graph()
 	
  def _generate_graph(self):
  """
  Membuat graf berdasarkan aturan yang telah dijelaskan.
  """
  for i in range(1, self.n + 1):
  for j in range(1, self.m + 1):
  self.V.add(f'x_{i},{j}')
  self.V.add(f'y_{i},{j}')
 	
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
 	
  for i in range(1, self.n + 1):
  for j in range(1, self.m):
  self.E.add((f'x_{i},1', f'x_{i},{j+1}'))
  self.E.add((f'y_{i},1', f'y_{i},{j+1}'))
	Pewarnaan ditemukan!
	Simpul dan Warnanya:
	x_1,1: 1
	x_1,2: 4
	x_1,3: 6
	x_1,4: 8
	x_2,1: 10
	x_2,2: 7
	x_2,3: 5
	x_2,4: 3
	y_1,1: 2
	y_1,2: 5
	y_1,3: 7
	y_1,4: 9
	y_2,1: 11
	y_2,2: 8
	y_2,3: 6
	y_2,4: 4
	Sisi dan Selisih Warnanya:
	x_1,1 - x_1,2: |1 - 4| = 3
	x_1,1 - x_1,3: |1 - 6| = 5
	x_1,1 - x_1,4: |1 - 8| = 7
	x_1,1 - x_2,1: |1 - 10| = 9
	x_1,1 - y_1,1: |1 - 2| = 1
	x_2,1 - x_2,2: |10 - 7| = 3
	x_2,1 - x_2,3: |10 - 5| = 5
	x_2,1 - x_2,4: |10 - 3| = 7
	x_2,1 - y_2,1: |10 - 11| = 1
	y_1,1 - y_1,2: |2 - 5| = 3
	y_1,1 - y_1,3: |2 - 7| = 5
	y_1,1 - y_1,4: |2 - 9| = 7
	y_1,1 - y_2,1: |2 - 11| = 9
	y_2,1 - y_2,2: |11 - 8| = 3
	y_2,1 - y_2,3: |11 - 6| = 5
	y_2,1 - y_2,4: |11 - 4| = 7
	
	Bilangan kromatik graceful ganjil (x_og): 11

```
        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in
self.color:
                edge_color = abs(self.color[u]
- self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self,
colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda dan
selisih ganjil.
        """
        self.color['x_1,1'] = 1
        self.color['y_1,1'] = 2
        self.color['x_2,1'] = 2 * m + 2
        self.color['y_2,1'] = 2 * m + 3

        # Menentukan warna untuk simpul
x_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_1,{j}'] = 2 * j

        # Menentukan warna untuk simpul
x_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_2,{j}'] = 2 * (m - j)
+ 3

        # Menentukan warna untuk simpul
y_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_1,{j}'] = 2 * j + 1
```

```
        # Menentukan warna untuk simpul
y_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_2,{j}'] = 2 * (m - j)
+ 4

        return self.color

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v,
"Belum diwarnai")}')

        print("\nSisi dan Selisih
Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in
self.color:
                print(f'{u} - {v}:
|{self.color[u]} - {self.color[v]}| =
{abs(self.color[u] - self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')

        # Menampilkan x_og (bilangan
kromatik graceful ganjil)
        used_colors =
len(set(self.color.values()))
        print(f"\nBilangan kromatik
graceful ganjil (x_og): {used_colors}")

# Contoh eksekusi untuk n=2, m=...
n=2
m=4
colors = [1, 2, 3, 4, 5, ..., 2 * m + 3]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
```

print("Tidak ditemukan pewarnaan yang valid.") graph.display()	

1. p. Pewarnaan pada graf $L_3 \rhd S_m, m \geq 2$

Kode	Output

class Graph:
def __init__(self, n, m):
self.n = n
self.m = m
self.V = set()
self.E = set()
self.color = {}
 	
# Generate graf 	
self._generate_graph()
 	
def _generate_graph(self):
"""
Membuat graf berdasarkan aturan yang telah dijelaskan.
"""
for i in range(1, self.n + 1):
for j in range(1, self.m + 1):
self.V.add(f'x_{i},{j}')
self.V.add(f'y_{i},{j}')
 	
for i in range(1, self.n):
self.E.add((f'x_{i},1', f'x_{i+1},1'))
self.E.add((f'y_{i},1', f'y_{i+1},1'))
 	
for i in range(1, self.n + 1):
for j in range(1, self.m):
self.E.add((f'x_{i},1', f'x_{i},{j+1}'))
self.E.add((f'y_{i},1', f'y_{i},{j+1}'))
 	
for i in range(1, self.n + 1):
self.E.add((f'x_{i},1', f'y_{i},1'))
 	
def is_valid_coloring(self):
"""
	Pewarnaan ditemukan!
	Simpul dan Warnanya:
	x_1,1: 3
	x_1,2: 4
	x_1,3: 6
	x_1,4: 8
	x_2,1: 12
	x_2,2: 9
	x_2,3: 7
	x_2,4: 5
	x_3,1: 11
	x_3,2: 8
	x_3,3: 6
	x_3,4: 4
	y_1,1: 10
	y_1,2: 9
	y_1,3: 7
	y_1,4: 5
	y_2,1: 1
	y_2,2: 4
	y_2,3: 6
	y_2,4: 8
	y_3,1: 2
	y_3,2: 5
	y_3,3: 7
	y_3,4: 9
	
	Sisi dan Selisih Warnanya:
	x_1,1 - x_1,2: |3 - 4| = 1
	x_1,1 - x_1,3: |3 - 6| = 3
	x_1,1 - x_1,4: |3 - 8| = 5
	x_1,1 - x_2,1: |3 - 12| = 9
	x_1,1 - y_1,1: |3 - 10| = 7
	x_2,1 - x_2,2: |12 - 9| = 3
	x_2,1 - x_2,3: |12 - 7| = 5
	x_2,1 - x_2,4: |12 - 5| = 7
	x_2,1 - x_3,1: |12 - 11| = 1
	x_2,1 - y_2,1: |12 - 1| = 11
	x_3,1 - x_3,2: |11 - 8| = 3
	x_3,1 - x_3,3: |11 - 6| = 5
	x_3,1 - x_3,4: |11 - 4| = 7
	x_3,1 - y_3,1: |11 - 2| = 9
	y_1,1 - y_1,2: |10 - 9| = 1
	y_1,1 - y_1,3: |10 - 7| = 3
	y_1,1 - y_1,4: |10 - 5| = 5
	y_1,1 - y_2,1: |10 - 1| = 9
	y_2,1 - y_2,2: |1 - 4| = 3
	y_2,1 - y_2,3: |1 - 6| = 5
	y_2,1 - y_2,4: |1 - 8| = 7
	y_2,1 - y_3,1: |1 - 2| = 1
	y_3,1 - y_3,2: |2 - 5| = 3
	y_3,1 - y_3,3: |2 - 7| = 5
	y_3,1 - y_3,4: |2 - 9| = 7
	
	Bilangan kromatik graceful ganjil (x_og): 12

```
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in
self.color:
                edge_color = abs(self.color[u]
- self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self,
colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda
dan selisih ganjil.
        """
        self.color['x_1,1'] = 3
        self.color['y_1,1'] = 2 * m + 2
        self.color['x_2,1'] = 2 * m + 4
        self.color['y_2,1'] = 1
        self.color['x_3,1'] = 2 * m + 3
        self.color['y_3,1'] = 2

        # Menentukan warna untuk simpul
x_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_1,{j}'] = 2 * j

        # Menentukan warna untuk simpul
x_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_2,{j}'] = 2 * (m - j)
+ 5

        # Menentukan warna untuk simpul
x_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_3,{j}'] = 2 * (m - j)
+ 4

        # Menentukan warna untuk simpul
y_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
```

```
            self.color[f'y_1,{j}'] = 2 * (m - j)
+ 5

        # Menentukan warna untuk simpul
y_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_2,{j}'] = 2 * j

        # Menentukan warna untuk simpul
y_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_3,{j}'] = 2 * j + 1

        return self.color

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v,
"Belum diwarnai")}')

        print("\nSisi dan Selisih
Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in
self.color:
                print(f'{u} - {v}:
|{self.color[u]} - {self.color[v]}| =
{abs(self.color[u] - self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')

        # Menampilkan x_og (bilangan
kromatik graceful ganjil)
        used_colors =
len(set(self.color.values()))
        print(f"\nBilangan kromatik
graceful ganjil (x_og): {used_colors}")

# Contoh eksekusi untuk n=3, m=...
n=3
```

```
m=4
colors = [1, 2, 3, 4, 5, ..., 2 * m + 4]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
    print("Tidak ditemukan pewarnaan
yang valid.")

graph.display()
```

1. q. Pewarnaan pada graf $L_4 \rhd S_m, m \geq 2$

Kode

```
class Graph:
    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.V = set()
        self.E = set()
        self.color = {}

        # Generate graf
        self._generate_graph()

    def _generate_graph(self):
        """
        Membuat graf berdasarkan aturan yang
telah dijelaskan.
        """
        for i in range(1, self.n + 1):
            for j in range(1, self.m + 1):
                self.V.add(f'x_{i},{j}')
                self.V.add(f'y_{i},{j}')

        for i in range(1, self.n):
            self.E.add((f'x_{i},1', f'x_{i+1},1'))
            self.E.add((f'y_{i},1', f'y_{i+1},1'))

        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},1',
f'x_{i},{j+1}'))
                self.E.add((f'y_{i},1',
f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
```

Output

```
Simpul dan Warnanya:
x_1,1: 3
x_1,2: 4
x_1,3: 6
x_1,4: 8
x_2,1: 12
x_2,2: 9
x_2,3: 7
x_2,4: 5
x_3,1: 13
x_3,2: 10
x_3,3: 8
x_3,4: 6
x_4,1: 4
x_4,2: 5
x_4,3: 7
x_4,4: 9
y_1,1: 10
y_1,2: 9
y_1,3: 7
y_1,4: 5
y_2,1: 1
y_2,2: 4
y_2,3: 6
y_2,4: 8
y_3,1: 2
y_3,2: 5
y_3,3: 7
y_3,4: 9
y_4,1: 11
y_4,2: 10
y_4,3: 8
y_4,4: 6
```

```
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda dan
selisih ganjil.
        """
        self.color['x_1,1'] = 3
        self.color['y_1,1'] = 2 * m + 2
        self.color['x_2,1'] = 2 * m + 4
        self.color['y_2,1'] = 1
        self.color['x_3,1'] = 2 * m + 5
        self.color['y_3,1'] = 2
        self.color['x_4,1'] = 4
        self.color['y_4,1'] = 2 * m + 3

        # Menentukan warna untuk simpul
x_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_1,{j}'] = 2 * j

        # Menentukan warna untuk simpul
x_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_2,{j}'] = 2 * (m - j) + 5

        # Menentukan warna untuk simpul
x_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_3,{j}'] = 2 * (m - j) + 6
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |3 - 4| = 1
x_1,1 - x_1,3: |3 - 6| = 3
x_1,1 - x_1,4: |3 - 8| = 5
x_1,1 - x_2,1: |3 - 12| = 9
x_1,1 - y_1,1: |3 - 10| = 7
x_2,1 - x_2,2: |12 - 9| = 3
x_2,1 - x_2,3: |12 - 7| = 5
x_2,1 - x_2,4: |12 - 5| = 7
x_2,1 - x_3,1: |12 - 13| = 1
x_2,1 - y_2,1: |12 - 1| = 11
x_3,1 - x_3,2: |13 - 10| = 3
x_3,1 - x_3,3: |13 - 8| = 5
x_3,1 - x_3,4: |13 - 6| = 7
x_3,1 - x_4,1: |13 - 4| = 9
x_3,1 - y_3,1: |13 - 2| = 11
x_4,1 - x_4,2: |4 - 5| = 1
x_4,1 - x_4,3: |4 - 7| = 3
x_4,1 - x_4,4: |4 - 9| = 5
x_4,1 - y_4,1: |4 - 11| = 7
y_1,1 - y_1,2: |10 - 9| = 1
y_1,1 - y_1,3: |10 - 7| = 3
y_1,1 - y_1,4: |10 - 5| = 5
y_1,1 - y_2,1: |10 - 1| = 9
y_2,1 - y_2,2: |1 - 4| = 3
y_2,1 - y_2,3: |1 - 6| = 5
y_2,1 - y_2,4: |1 - 8| = 7
y_2,1 - y_3,1: |1 - 2| = 1
y_3,1 - y_3,2: |2 - 5| = 3
y_3,1 - y_3,3: |2 - 7| = 5
y_3,1 - y_3,4: |2 - 9| = 7
y_3,1 - y_4,1: |2 - 11| = 9
y_4,1 - y_4,2: |11 - 10| = 1
y_4,1 - y_4,3: |11 - 8| = 3
y_4,1 - y_4,4: |11 - 6| = 5

Bilangan kromatik graceful ganjil (x_og): 13
```

```
        # Menentukan warna untuk simpul
x_4,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_4,{j}'] = 2 * j + 1

        # Menentukan warna untuk simpul
y_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_1,{j}'] = 2 * (m - j) + 5

        # Menentukan warna untuk simpul
y_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_2,{j}'] = 2 * j

        # Menentukan warna untuk simpul
y_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_3,{j}'] = 2 * j + 1

        # Menentukan warna untuk simpul
y_4,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_4,{j}'] = 2 * (m - j) + 6

        return self.color

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v,
"Belum diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')
```

```
        # Menampilkan x_og (bilangan
kromatik graceful ganjil)
        used_colors =
len(set(self.color.values()))
        print(f"\nBilangan kromatik graceful
ganjil (x_og): {used_colors}")

# Contoh eksekusi untuk n=4, m=...
n=4
m=4
colors = [1, 2, 3, 4, 5, ..., 2 * m + 5]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
    print("Tidak ditemukan pewarnaan yang
valid.")

graph.display()
```

1. r. Pewarnaan pada graf $L_5 \rhd S_m, m \geq 2$

Kode	Output

class Graph:
  def __init__(self, n, m):
  self.n = n
  self.m = m
  self.V = set()
  self.E = set()
  self.color = {}
 	
  # Generate graf
  self._generate_graph()
 	
  def _generate_graph(self):
  """
  Membuat graf berdasarkan aturan yang telah dijelaskan.
  """
  for i in range(1, self.n + 1):
  for j in range(1, self.m + 1):
  self.V.add(f'x_{i},{j}')
  self.V.add(f'y_{i},{j}')
 	
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
	Simpul dan Warnanya:
	x_1,1: 2
	x_1,2: 5
	x_1,3: 7
	x_2,1: 3
	x_2,2: 6
	x_2,3: 8
	x_3,1: 12
	x_3,2: 9
	x_3,3: 7
	x_4,1: 11
	x_4,2: 8
	x_4,3: 6
	x_5,1: 4
	x_5,2: 5
	x_5,3: 7
	y_1,1: 9
	y_1,2: 6
	y_1,3: 4
	y_2,1: 10
	y_2,2: 7
	y_2,3: 5
	y_3,1: 1
	y_3,2: 4
	y_3,3: 6
	y_4,1: 2
	y_4,2: 5
	y_4,3: 7
	y_5,1: 9
	y_5,2: 8
	y_5,3: 6

```
        for i in range(1, self.n + 1):
            for j in range(1, self.m):
                self.E.add((f'x_{i},1',
f'x_{i},{j+1}'))
                self.E.add((f'y_{i},1',
f'y_{i},{j+1}'))

        for i in range(1, self.n + 1):
            self.E.add((f'x_{i},1', f'y_{i},1'))

    def is_valid_coloring(self):
        """
        Memeriksa apakah pewarnaan
memenuhi syarat graceful ganjil.
        """
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
self.color[u] == self.color[v]:
                    return False
        return True

    def graceful_odd_coloring(self, colors):
        """
        Mencari pewarnaan graceful ganjil
dengan memastikan bahwa simpul
bertetangga memiliki warna berbeda dan
selisih ganjil.
        """
        self.color['x_1,1'] = 2
        self.color['y_1,1'] = 2 * m + 3
        self.color['x_2,1'] = 3
        self.color['y_2,1'] = 2 * m + 4
        self.color['x_3,1'] = 2 * m + 6
        self.color['y_3,1'] = 1
        self.color['x_4,1'] = 2 * m + 5
        self.color['y_4,1'] = 2
        self.color['x_5,1'] = 4
        self.color['y_5,1'] = 2 * m + 3

        # Menentukan warna untuk simpul
x_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_1,{j}'] = 2 * j + 1
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |2 - 5| = 3
x_1,1 - x_1,3: |2 - 7| = 5
x_1,1 - x_2,1: |2 - 3| = 1
x_1,1 - y_1,1: |2 - 9| = 7
x_2,1 - x_2,2: |3 - 6| = 3
x_2,1 - x_2,3: |3 - 8| = 5
x_2,1 - x_3,1: |3 - 12| = 9
x_2,1 - y_2,1: |3 - 10| = 7
x_3,1 - x_3,2: |12 - 9| = 3
x_3,1 - x_3,3: |12 - 7| = 5
x_3,1 - x_4,1: |12 - 11| = 1
x_3,1 - y_3,1: |12 - 1| = 11
x_4,1 - x_4,2: |11 - 8| = 3
x_4,1 - x_4,3: |11 - 6| = 5
x_4,1 - x_5,1: |11 - 4| = 7
x_4,1 - y_4,1: |11 - 2| = 9
x_5,1 - x_5,2: |4 - 5| = 1
x_5,1 - x_5,3: |4 - 7| = 3
x_5,1 - y_5,1: |4 - 9| = 5
y_1,1 - y_1,2: |9 - 6| = 3
y_1,1 - y_1,3: |9 - 4| = 5
y_1,1 - y_2,1: |9 - 10| = 1
y_2,1 - y_2,2: |10 - 7| = 3
y_2,1 - y_2,3: |10 - 5| = 5
y_2,1 - y_3,1: |10 - 1| = 9
y_3,1 - y_3,2: |1 - 4| = 3
y_3,1 - y_3,3: |1 - 6| = 5
y_3,1 - y_4,1: |1 - 2| = 1
y_4,1 - y_4,2: |2 - 5| = 3
y_4,1 - y_4,3: |2 - 7| = 5
y_4,1 - y_5,1: |2 - 9| = 7
y_5,1 - y_5,2: |9 - 8| = 1
y_5,1 - y_5,3: |9 - 6| = 3

Bilangan kromatik graceful ganjil (x_og): 12
```

```
        # Menentukan warna untuk simpul
x_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_2,{j}'] = 2 * j + 2

        # Menentukan warna untuk simpul
x_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_3,{j}'] = 2 * (m - j) +
7

        # Menentukan warna untuk simpul
x_4,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_4,{j}'] = 2 * (m - j) +
6

        # Menentukan warna untuk simpul
x_5,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'x_5,{j}'] = 2 * j + 1

        # Menentukan warna untuk simpul
y_1,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_1,{j}'] = 2 * (m - j) +
4

        # Menentukan warna untuk simpul
y_2,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_2,{j}'] = 2 * (m - j) +
5

        # Menentukan warna untuk simpul
y_3,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_3,{j}'] = 2 * j

        # Menentukan warna untuk simpul
y_4,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
            self.color[f'y_4,{j}'] = 2 * j + 1

        # Menentukan warna untuk simpul
y_5,j dengan 2 ≤ j ≤ m
        for j in range(2, m + 1):
```

```
            self.color[f'y_5,{j}'] = 2 * (m - j) +
6

        return self.color

        return self.is_valid_coloring()

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v,
"Belum diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]}
- {self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum
diwarnai')

        # Menampilkan x_og (bilangan
kromatik graceful ganjil)
        used_colors =
len(set(self.color.values()))
        print(f"\nBilangan kromatik graceful
ganjil (x_og): {used_colors}")

# Contoh eksekusi untuk n=5, m=...
n=5
m=3
colors = [1, 2, 3, 4, 5, ..., 2 * m + 6]
graph = Graph(n, m)
if graph.graceful_odd_coloring(colors):
    print("Pewarnaan ditemukan!")
else:
    print("Tidak ditemukan pewarnaan yang
valid.")

graph.display()
```

1. s. Pewarnaan pada graf $L_n \rhd S_m, n \geq 6, m \geq 2$

Kode	Output

class Graph:
  def __init__(self, n, m):
  self.n = n
  self.m = m
  self.V = set()
  self.E = set()
  self.color = {}
 	
  # Generate graf
  self._generate_graph()
 	
  def _generate_graph(self):
  """
  Membuat graf berdasarkan aturan yang telah dijelaskan.
  """
  for i in range(1, self.n + 1):
  for j in range(1, self.m + 1):
  self.V.add(f'x_{i},{j}')
  self.V.add(f'y_{i},{j}')
 	
  for i in range(1, self.n):
  self.E.add((f'x_{i},1', f'x_{i+1},1'))
  self.E.add((f'y_{i},1', f'y_{i+1},1'))
 	
  for i in range(1, self.n + 1):
  for j in range(1, self.m):
  self.E.add((f'x_{i},1', f'x_{i},{j+1}'))
  self.E.add((f'y_{i},1', f'y_{i},{j+1}'))
 	
  for i in range(1, self.n + 1):
  self.E.add((f'x_{i},1', f'y_{i},1'))
 	
  def apply_coloring_rule(self):
  """
  Menerapkan aturan pewarnaan graceful ganjil.
  """
  # Pewarnaan untuk simpul x_(i,j) dan y_(i,j)
  for i in range(1, self.n + 1):
  for j in range(1, self.m + 1):
  # Pewarnaan untuk x_(i,j)
	Simpul dan Warnanya:
	x_1,1: 12
	x_1,2: 9
	x_1,3: 7
	x_1,4: 5
	x_2,1: 3
	x_2,2: 6
	x_2,3: 8
	x_2,4: 10
	x_3,1: 14
	x_3,2: 11
	x_3,3: 9
	x_3,4: 7
	x_4,1: 1
	x_4,2: 4
	x_4,3: 6
	x_4,4: 8
	x_5,1: 12
	x_5,2: 9
	x_5,3: 7
	x_5,4: 5
	x_6,1: 3
	x_6,2: 6
	x_6,3: 8
	x_6,4: 10
	y_1,1: 13
	y_1,2: 10
	y_1,3: 8
	y_1,4: 6
	y_2,1: 4
	y_2,2: 7
	y_2,3: 9
	y_2,4: 11
	y_3,1: 15
	y_3,2: 12
	y_3,3: 10
	y_3,4: 8
	y_4,1: 2
	y_4,2: 4
	y_4,3: 6
	y_4,4: 8
	y_5,1: 13
	y_5,2: 10
	y_5,3: 8
	y_5,4: 6
	y_6,1: 4
	y_6,2: 7
	y_6,3: 9
	y_6,4: 11

```
            if i % 4 == 0 and j == 1:
                self.color[f'x_{i},{j}'] = 1
            elif i % 4 == 2 and j == 1:
                self.color[f'x_{i},{j}'] = 3
            elif i % 4 == 1 and j == 1:
                self.color[f'x_{i},{j}'] = 2 *
self.m + 4
            elif i % 4 == 3 and j == 1:
                self.color[f'x_{i},{j}'] = 2 *
self.m + 6
            elif i % 4 == 1:
                self.color[f'x_{i},{j}'] = 2 *
(self.m - j) + 5
            elif i % 4 == 2:
                self.color[f'x_{i},{j}'] = 2 * j +
2
            elif i % 4 == 3:
                self.color[f'x_{i},{j}'] = 2 *
(self.m - j) + 7
            elif i % 4 == 0:
                self.color[f'x_{i},{j}'] = 2 * j

            # Pewarnaan untuk y_(i,j)
            if i % 4 == 0 and j == 1:
                self.color[f'y_{i},{j}'] = 2
            elif i % 4 == 1 and j == 1:
                self.color[f'y_{i},{j}'] = 2 *
self.m + 5
            elif i % 4 == 2 and j == 1:
                self.color[f'y_{i},{j}'] = 4
            elif i % 4 == 3 and j == 1:
                self.color[f'y_{i},{j}'] = 2 *
self.m + 7
            elif i % 4 == 0:
                self.color[f'y_{i},{j}'] = 2 * j
            elif i % 4 == 1:
                self.color[f'y_{i},{j}'] = 2 *
(self.m - j) + 6
            elif i % 4 == 2:
                self.color[f'y_{i},{j}'] = 2 * j +
3
            elif i % 4 == 3:
                self.color[f'y_{i},{j}'] = 2 *
(self.m - j) + 8

    def graceful_odd_coloring(self):
        """
```

```
Sisi dan Selisih Warnanya:
x_1,1 - x_1,2: |12 - 9| = 3
x_1,1 - x_1,3: |12 - 7| = 5
x_1,1 - x_1,4: |12 - 5| = 7
x_1,1 - x_2,1: |12 - 3| = 9
x_1,1 - y_1,1: |12 - 13| = 1
x_2,1 - x_2,2: |3 - 6| = 3
x_2,1 - x_2,3: |3 - 8| = 5
x_2,1 - x_2,4: |3 - 10| = 7
x_2,1 - x_3,1: |3 - 14| = 11
x_2,1 - y_2,1: |3 - 4| = 1
x_3,1 - x_3,2: |14 - 11| = 3
x_3,1 - x_3,3: |14 - 9| = 5
x_3,1 - x_3,4: |14 - 7| = 7
x_3,1 - x_4,1: |14 - 1| = 13
x_3,1 - y_3,1: |14 - 15| = 1
x_4,1 - x_4,2: |1 - 4| = 3
x_4,1 - x_4,3: |1 - 6| = 5
x_4,1 - x_4,4: |1 - 8| = 7
x_4,1 - x_5,1: |1 - 12| = 11
x_4,1 - y_4,1: |1 - 2| = 1
x_5,1 - x_5,2: |12 - 9| = 3
x_5,1 - x_5,3: |12 - 7| = 5
x_5,1 - x_5,4: |12 - 5| = 7
x_5,1 - x_6,1: |12 - 3| = 9
x_5,1 - y_5,1: |12 - 13| = 1
x_6,1 - x_6,2: |3 - 6| = 3
x_6,1 - x_6,3: |3 - 8| = 5
x_6,1 - x_6,4: |3 - 10| = 7
x_6,1 - y_6,1: |3 - 4| = 1
y_1,1 - y_1,3: |13 - 8| = 5
y_1,1 - y_1,4: |13 - 6| = 7
y_1,1 - y_2,1: |13 - 4| = 9
y_2,1 - y_2,2: |4 - 7| = 3
y_2,1 - y_2,3: |4 - 9| = 5
y_2,1 - y_2,4: |4 - 11| = 7
y_2,1 - y_3,1: |4 - 15| = 11
y_3,1 - y_3,2: |15 - 12| = 3
y_3,1 - y_3,3: |15 - 10| = 5
y_3,1 - y_3,4: |15 - 8| = 7
y_3,1 - y_4,1: |15 - 2| = 13
y_4,1 - y_4,2: |2 - 4| = 2
y_4,1 - y_4,3: |2 - 6| = 4
y_4,1 - y_4,4: |2 - 8| = 6
y_4,1 - y_5,1: |2 - 13| = 11
y_5,1 - y_5,2: |13 - 10| = 3
y_5,1 - y_5,3: |13 - 8| = 5
y_5,1 - y_5,4: |13 - 6| = 7
y_5,1 - y_6,1: |13 - 4| = 9
y_6,1 - y_6,2: |4 - 7| = 3
y_6,1 - y_6,3: |4 - 9| = 5
y_6,1 - y_6,4: |4 - 11| = 7
```

```
        Mengecek apakah pewarnaan
memenuhi kondisi graceful ganjil.
        """
        edge_colors = set()
        for u, v in self.E:
            if u in self.color and v in self.color:
                edge_color = abs(self.color[u] -
self.color[v])
                if edge_color % 2 == 0 or
edge_color in edge_colors:
                    return False
                edge_colors.add(edge_color)
        return True

    def display(self):
        """
        Menampilkan simpul, sisi, dan
warnanya.
        """
        print("Simpul dan Warnanya:")
        for v in sorted(self.V):
            print(f'{v}: {self.color.get(v, "Belum
diwarnai")}')

        print("\nSisi dan Selisih Warnanya:")
        for u, v in sorted(self.E):
            if u in self.color and v in self.color:
                print(f'{u} - {v}: |{self.color[u]} -
{self.color[v]}| = {abs(self.color[u] -
self.color[v])}')
            else:
                print(f'{u} - {v}: Belum diwarnai')

# Contoh
n = 6
m = 4
graph = Graph(n, m)
graph.apply_coloring_rule()
graph.display()
```

Lampiran 2. Riwayat Hidup

RIWAYAT HIDUP

Ni Nyoman Trisya Inne Kristhina lahir di Denpasar pada tanggal 25 Mei 2003. Penulis lahir dari pasangan suami istri Bapak I Gede Mangku Astawa dan Ibu Ni Wayan Sariati berkebangsaan Indonesia dan beragama Hindu. Kini penulis beralamat di Banjar Dinas Taman Sari, Padangbulia, Kecamatan Sukasada, Kabupaten Buleleng, Provinsi Bali. Penulis menyelesaikan pendidikan dasar di SD K Soverdi Tuban dan lulus pada tahun 2015. Kemudian penulis melanjutkan di SMP K Soverdi Tuban dan lulus pada tahun 2018. Pada tahun 2021, penulis lulus dari SMA Negeri 4 Singaraja dan melanjutkan ke Strata 1 Jurusan Matematika di Universitas Pendidikan Ganesha. Pada semester akhir tahun 2025 penulis telah menyelesaikan Tugas Akhir yang berjudul "Bilangan Kromatik Graceful Ganjil pada Graf Hasil Operasi *Comb* Graf Tangga". Selanjutnya, mulai tahun 2021 sampai dengan penulisan skripsi ini, penulis masih terdaftar sebagai mahasiswa Program Studi S1 Matematika di Universitas Pendidikan Ganesha.