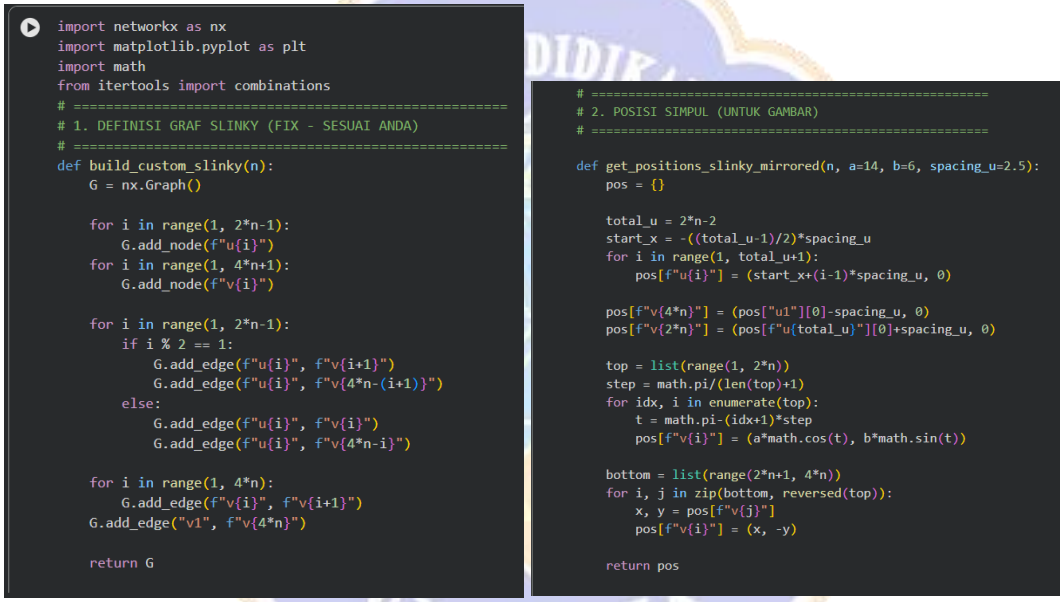
DAFTAR LAMPIRAN

Lampiran 1. Program Lintasan pelangi dari pewarnaan antiajaib pelangi pada kasus $n \geq 3$.

Bisa di lihat pada link ini

https://colab.research.google.com/drive/1SQAJaXvJJVQARwIkzfdVmmPfNZjCBQH3?usp=sharing



```python
import networkx as nx
import matplotlib.pyplot as plt
import math
from itertools import combinations
# ==========================================
# 1. DEFINISI GRAF SLINKY (FIX - SESUAI ANDA)
# ==========================================
def build_custom_slinky(n):
    G = nx.Graph()

    for i in range(1, 2*n-1):
        G.add_node(f"u{i}")
    for i in range(1, 4*n+1):
        G.add_node(f"v{i}")

    for i in range(1, 2*n-1):
        if i % 2 == 1:
            G.add_edge(f"u{i}", f"v{i+1}")
            G.add_edge(f"u{i}", f"v{4*n-(i+1)}")
        else:
            G.add_edge(f"u{i}", f"v{i}")
            G.add_edge(f"u{i}", f"v{4*n-i}")

    for i in range(1, 4*n):
        G.add_edge(f"v{i}", f"v{i+1}")
    G.add_edge("v1", f"v{4*n}")

    return G
```

```python
# ==========================================
# 2. POSISI SIMPUL (UNTUK GAMBAR)
# ==========================================

def get_positions_slinky_mirrored(n, a=14, b=6, spacing_u=2.5):
    pos = {}

    total_u = 2*n-2
    start_x = -((total_u-1)/2)*spacing_u
    for i in range(1, total_u+1):
        pos[f"u{i}"] = (start_x+(i-1)*spacing_u, 0)

    pos[f"v{4*n}"] = (pos["u1"][0]-spacing_u, 0)
    pos[f"v{2*n}"] = (pos[f"u{total_u}"][0]+spacing_u, 0)

    top = list(range(1, 2*n))
    step = math.pi/(len(top)+1)
    for idx, i in enumerate(top):
        t = math.pi-(idx+1)*step
        pos[f"v{i}"] = (a*math.cos(t), b*math.sin(t))

    bottom = list(range(2*n+1, 4*n))
    for i, j in zip(bottom, reversed(top)):
        x, y = pos[f"v{j}"]
        pos[f"v{i}"] = (x, -y)

    return pos
```

```python
# ======================================================
# 3. PEWARNAAN SISI SESUAI RUMUS ANDA
# ======================================================
def slinky_edge_weighting(G, n):
    w = {}
    # w(v1, v4n) = 1
    w[("v1", f"v{4*n}")] = 1
    # w(v_i, v_{i+1})
    for i in range(1, 4*n):
        if i <= 2*n-1:
            w[(f"v{i}", f"v{i+1}")] = i+1
        else:
            w[(f"v{i}", f"v{i+1}")] = i+1-2*n
    # u_i - v_(i+1) dan u_i - v_(4n-(i+1)) (i ganjil)
    for i in range(1, 2*n-1, 2):
        if i <= n-1:
            val = 2*n-(i-1)
        else:
            val = i+2
        w[(f"u{i}", f"v{i+1}")] = val
        w[(f"u{i}", f"v{4*n-(i+1)}")] = val
    # u_i - v_i dan u_i - v_(4n-i) (i genap)
    for i in range(2, 2*n-1, 2):
        if i <= n:
            val = i-1
        else:
            val = 2*n-i
        w[(f"u{i}", f"v{i}")] = val
        w[(f"u{i}", f"v{4*n-i}")] = val
    return w
```

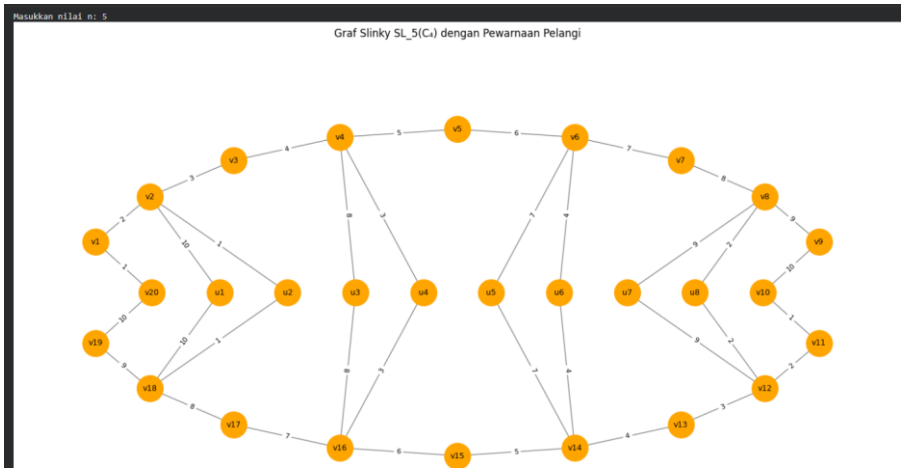
```python
# ======================================================
# 4. CEK LINTASAN PELANGI
# ======================================================
def is_rainbow_path(path, edge_weight):
    used = set()
    for i in range(len(path)-1):
        e = (path[i], path[i+1])
        if e not in edge_weight:
            e = (path[i+1], path[i])
        if edge_weight[e] in used:
            return False
        used.add(edge_weight[e])
    return True
def find_rainbow_path(G, s, t, edge_weight, depth=12):
    def dfs(cur, target, visited, used, path):
        if cur == target:
            return path
        if len(path) > depth:
            return None
        for nb in G.neighbors(cur):
            e = (cur, nb)
            if e not in edge_weight:
                e = (nb, cur)
            w = edge_weight[e]
            if nb not in visited and w not in used:
                res = dfs(nb, target,
                          visited | {nb},
                          used | {w},
                          path + [nb])
                if res:
                    return res
        return None
    return dfs(s, t, {s}, set(), [s])
```

Program ini untuk mengecek semua lintasan pelangi

```python
n = int(input("Masukkan nilai n: "))

G = build_custom_slinky(n)
pos = get_positions_slinky_mirrored(n)
edge_weight = slinky_edge_weighting(G, n)

# Gambar graf
plt.figure(figsize=(14, 8))
nx.draw(G, pos, with_labels=True,
        node_color="orange", node_size=900,
        font_size=9, edge_color="gray")
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_weight, font_size=8)
plt.title(f"Graf Slinky SL_{n}(C₄) dengan Pewarnaan Pelangi")
plt.axis("equal")
plt.show()

# Tampilkan lintasan pelangi
print("\nLINTASAN PELANGI UNTUK SETIAP PASANGAN SIMPUL:\n")
for u, v in combinations(G.nodes(), 2):
    path = find_rainbow_path(G, u, v, edge_weight)
    if path:
        print(f"{u} → {v} : {path}")
    else:
        print(f"{u} → {v} : TIDAK DITEMUKAN")
```

```
Masukkan nilai n: 5
```

Output




```
LINTASAN PELANGI UNTUK SETIAP PASANGAN SIMPUL:

u1 → u2 : ['u1', 'v2', 'u2']
u1 → u3 : ['u1', 'v2', 'v3', 'v4', 'u3']
u1 → u4 : ['u1', 'v18', 'v17', 'v16', 'u4']
u1 → u5 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'u5']
u1 → u6 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14', 'u6']
u1 → u7 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'u7']
u1 → u8 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'u8']
u1 → v1 : ['u1', 'v2', 'v1']
u1 → v2 : ['u1', 'v2']
u1 → v3 : ['u1', 'v2', 'v3']
u1 → v4 : ['u1', 'v2', 'v3', 'v4']
u1 → v5 : ['u1', 'v2', 'v3', 'v4', 'v5']
u1 → v6 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6']
u1 → v7 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7']
u1 → v8 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8']
u1 → v9 : ['u1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9']
u1 → v10 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14', 'v13', 'v12', 'v11', 'v1
u1 → v11 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14', 'v13', 'v12', 'v11']
u1 → v12 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14', 'v13', 'v12']
u1 → v13 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14', 'v13']
u1 → v14 : ['u1', 'v18', 'v17', 'v16', 'v15', 'v14']
u1 → v15 : ['u1', 'v18', 'v17', 'v16', 'v15']
u1 → v16 : ['u1', 'v18', 'v17', 'v16']
u1 → v17 : ['u1', 'v18', 'v17']
u1 → v18 : ['u1', 'v18']
u1 → v19 : ['u1', 'v18', 'v19']
u1 → v20 : ['u1', 'v2', 'v1', 'v20']
u2 → u3 : ['u2', 'v2', 'v3', 'v4', 'u3']
u2 → u4 : ['u2', 'v18', 'v17', 'v16', 'u4']
u2 → u5 : ['u2', 'v2', 'v3', 'v4', 'v5', 'v6', 'u5']
u2 → u6 : ['u2', 'v18', 'v17', 'v16', 'v15', 'v14', 'u6']
u2 → u7 : ['u2', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'u7']
u2 → u8 : ['u2', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'u8']
u2 → v1 : ['u2', 'v2', 'v1']
u2 → v2 : ['u2', 'v2']
u2 → v3 : ['u2', 'v2', 'v3']
u2 → v4 : ['u2', 'v2', 'v3', 'v4']
u2 → v5 : ['u2', 'v2', 'v3', 'v4', 'v5']
u2 → v6 : ['u2', 'v2', 'v3', 'v4', 'v5', 'v6']
```

```
    v12 → v16 : ['v12', 'v13', 'v14', 'v15', 'v16']
    v12 → v17 : ['v12', 'v13', 'v14', 'v15', 'v16', 'v17']
...  v12 → v18 : ['v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18']
    v12 → v19 : ['v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19']
    v12 → v20 : ['v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v20']
    v13 → v14 : ['v13', 'v14']
    v13 → v15 : ['v13', 'v14', 'v15']
    v13 → v16 : ['v13', 'v14', 'v15', 'v16']
    v13 → v17 : ['v13', 'v14', 'v15', 'v16', 'v17']
    v13 → v18 : ['v13', 'v14', 'v15', 'v16', 'v17', 'v18']
    v13 → v19 : ['v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19']
    v13 → v20 : ['v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v20']
    v14 → v15 : ['v14', 'v15']
    v14 → v16 : ['v14', 'v15', 'v16']
    v14 → v17 : ['v14', 'v15', 'v16', 'v17']
    v14 → v18 : ['v14', 'v15', 'v16', 'v17', 'v18']
    v14 → v19 : ['v14', 'v15', 'v16', 'v17', 'v18', 'v19']
    v14 → v20 : ['v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v20']
    v15 → v16 : ['v15', 'v16']
    v15 → v17 : ['v15', 'v16', 'v17']
    v15 → v18 : ['v15', 'v16', 'v17', 'v18']
    v15 → v19 : ['v15', 'v16', 'v17', 'v18', 'v19']
    v15 → v20 : ['v15', 'v16', 'v17', 'v18', 'v19', 'v20']
    v16 → v17 : ['v16', 'v17']
    v16 → v18 : ['v16', 'v17', 'v18']
    v16 → v19 : ['v16', 'v17', 'v18', 'v19']
    v16 → v20 : ['v16', 'v17', 'v18', 'v19', 'v20']
    v17 → v18 : ['v17', 'v18']
    v17 → v19 : ['v17', 'v18', 'v19']
    v17 → v20 : ['v17', 'v18', 'v19', 'v20']
    v18 → v19 : ['v18', 'v19']
    v18 → v20 : ['v18', 'v19', 'v20']
    v19 → v20 : ['v19', 'v20']
```

Program ini untuk mengecek lintasan pelangi dari titik awal ketitik tujuan

```
# ====================================================
# 5. MAIN PROGRAM
# ====================================================

def draw_graph_with_path(G, pos, edge_weight, path=None):
    plt.figure(figsize=(12,6))

    # semua edge
    nx.draw_networkx_edges(G, pos, width=1.5, alpha=0.5)

    # highlight lintasan pelangi
    if path:
        path_edges = [(path[i], path[i+1]) for i in range(len(path)-1)]
        nx.draw_networkx_edges(
            G, pos,
            edgelist=path_edges,
            width=4
        )

    # node
    nx.draw_networkx_nodes(G, pos, node_size=500)
    nx.draw_networkx_labels(G, pos, font_size=10)

    # label bobot sisi
    edge_labels = {}
    for (u,v), w in edge_weight.items():
        edge_labels[(u,v)] = w

    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)

    plt.title("Graf Slinky dengan Lintasan Pelangi")
    plt.axis("off")
    plt.show()
```

```
# ====================================================
# CARI LINTASAN PELANGI TERDEKAT
# ====================================================

def shortest_rainbow_path(G, s, t, edge_weight, max_depth=20):
    best_path = None

    for d in range(2, max_depth+1):
        path = find_rainbow_path(G, s, t, edge_weight, depth=d)
        if path:
            best_path = path
            break

    return best_path
```

```python
# ========================================================
# PROGRAM UTAMA
# ========================================================

if __name__ == "__main__":

    # input n
    n = int(input("Masukkan nilai n : "))

    # bangun graf
    G = build_custom_slinky(n)
    pos = get_positions_slinky_mirrored(n)
    edge_weight = slinky_edge_weighting(G, n)

    # tampilkan simpul
    print("\nDaftar titik pada graf:")
    print(sorted(G.nodes()))

    # input titik
    start = input("\nMasukkan titik awal : ")
    target = input("Masukkan titik tujuan : ")

    if start not in G.nodes() or target not in G.nodes():
        print("Titik tidak valid!")
        exit()

    # cari lintasan pelangi
    path = shortest_rainbow_path(G, start, target, edge_weight)

    if path:
        print("\nLintasan pelangi terdekat ditemukan:")
        print(" -> ".join(path))
    else:
        print("\nTidak ditemukan lintasan pelangi.")

    # gambar graf
    draw_graph_with_path(G, pos, edge_weight, path)
```

Output seperti ini

```
Masukkan nilai n : 5

Daftar titik pada graf:
['u1', 'u2', 'u3', 'u4', 'u5', 'u6', 'u7', 'u8', 'v1', 'v10', 'v11', 'v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v2', 'v20', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9']

Masukkan titik awal : u4
Masukkan titik tujuan : u1

Lintasan pelangi terdekat ditemukan:
u4 -> v16 -> v17 -> v18 -> u1
```