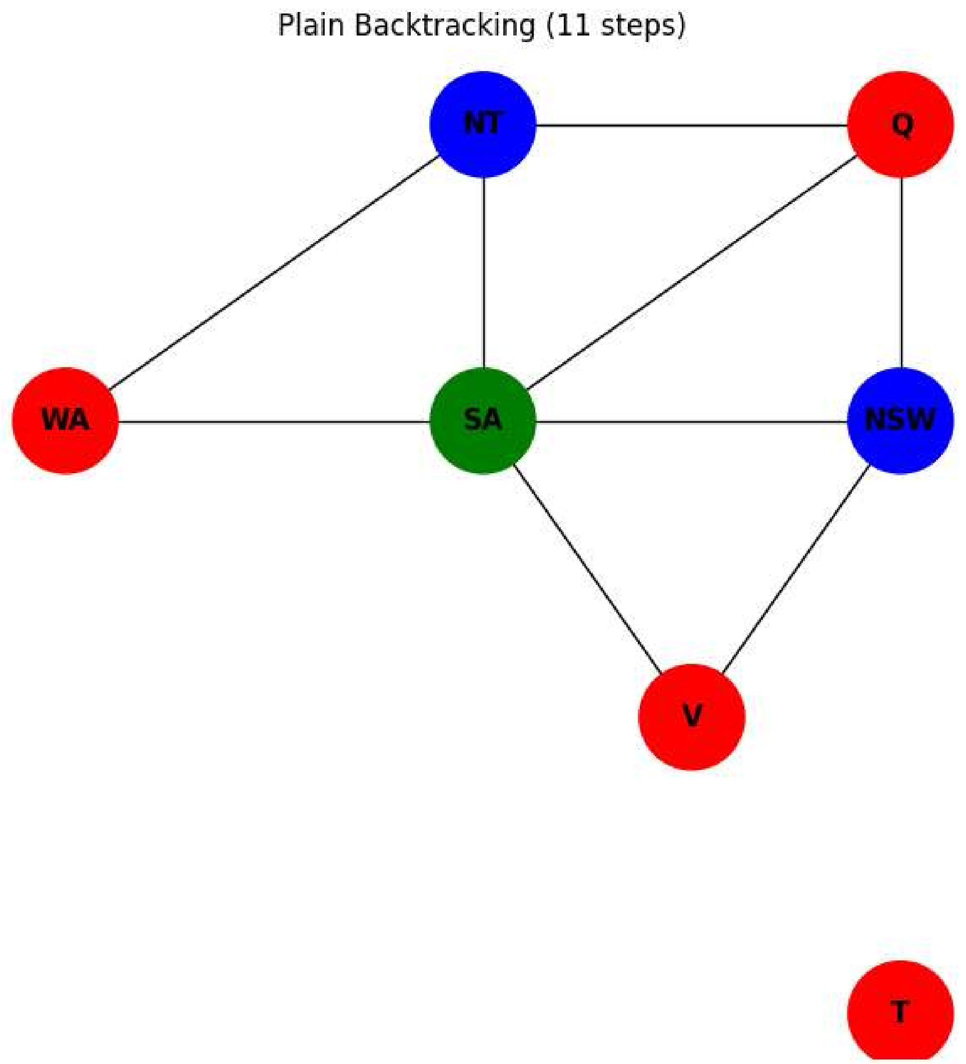
```
# Map Coloring Problem using Backtracking
# Demo for Part A: Plain Backtracking vs Backtracking with MRV + LCV + Visualization

import matplotlib.pyplot as plt
import networkx as nx

variables = ["V", "SA", "NT", "WA", "NSW", "Q", "T"]
colors = ["Red", "Green", "Blue"]

# Adjacency list (neighbors)
neighbors = {
    "WA": ["NT", "SA"],
    "NT": ["WA", "SA", "Q"],
    "SA": ["WA", "NT", "Q", "NSW", "V"],
    "Q": ["NT", "SA", "NSW"],
    "NSW": ["SA", "Q", "V"],
    "V": ["SA", "NSW"],
    "T": []
}


# ------------------ Utility Functions ------------------

# Check if assignment is valid
def is_valid(assignment, var, value):
    for n in neighbors[var]:
        if n in assignment and assignment[n] == value:
            return False
    return True


# ------------------ Plain Backtracking ------------------

plain_steps = 0
def backtrack_plain(assignment):
    global plain_steps
    plain_steps += 1

    if len(assignment) == len(variables):
        return assignment

    # Pick first unassigned variable (no heuristic)
    for var in variables:
        if var not in assignment:
            break

    for value in colors:  # try values in fixed order
        if is_valid(assignment, var, value):
            assignment[var] = value
            result = backtrack_plain(assignment)
            if result:
                return result
            del assignment[var]  # backtrack

    return None





# ------------------ Visualization ------------------

def visualize_coloring(solution, title):
    G = nx.Graph()
```

```
    # Add nodes and edges
    for var in variables:
        G.add_node(var)
    for var, neighs in neighbors.items():
        for n in neighs:
            G.add_edge(var, n)

    # Node positions for Australia-like layout
    pos = {
        "WA": (0, 0),
        "NT": (2, 2),
        "SA": (2, 0),
        "Q": (4, 2),
        "NSW": (4, 0),
        "V": (3, -2),
        "T": (4, -4)
    }

    node_colors = [solution.get(node, "white") for node in G.nodes()]

    plt.figure(figsize=(6, 6))
    nx.draw(
        G, pos,
        with_labels=True,
        node_color=node_colors,
        node_size=2000,
        font_size=12,
        font_weight="bold",
        edge_color="black"
    )
    plt.title(title)
    plt.show()


# ------------------ Run Demo ------------------
if __name__ == "__main__":
    # Plain Backtracking
    plain_solution = backtrack_plain({})
    print("Plain Backtracking Solution:", plain_solution)
    print("Plain Backtracking Steps:", plain_steps)
    visualize_coloring(plain_solution, f"Plain Backtracking ({plain_steps} steps)")
```

```
Plain Backtracking Solution: {'V': 'Red', 'SA': 'Green', 'NT': 'Blue', 'WA': 'Red', 'NSW': 'Blue', 'Q': 'Red', 'T': 'Red'}
Plain Backtracking Steps: 11
```




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

variables = ["V", "SA", "NT", "WA", "NSW", "Q", "T"]
colors = ["Red", "Green", "Blue"]

# Adjacency list (neighbors)
neighbors = {
    "WA": ["NT", "SA"],
    "NT": ["WA", "SA", "Q"],
    "SA": ["WA", "NT", "Q", "NSW", "V"],
    "Q": ["NT", "SA", "NSW"],
    "NSW": ["SA", "Q", "V"],
    "V": ["SA", "NSW"],
    "T": []
}


# ------------------ Utility Functions ------------------

# Check if assignment is valid
def is_valid(assignment, var, value):
    for n in neighbors[var]:
        if n in assignment and assignment[n] == value:
            return False
    return True
```

```
# ------------------ Visualization ------------------

def visualize_coloring(solution, title):
    G = nx.Graph()

    # Add nodes and edges
    for var in variables:
        G.add_node(var)
    for var, neighs in neighbors.items():
        for n in neighs:
            G.add_edge(var, n)

    # Node positions for Australia-like layout
    pos = {
        "WA": (0, 0),
        "NT": (2, 2),
        "SA": (2, 0),
        "Q": (4, 2),
        "NSW": (4, 0),
        "V": (3, -2),
        "T": (4, -4)
    }

    node_colors = [solution.get(node, "white") for node in G.nodes()]

    plt.figure(figsize=(6, 6))
    nx.draw(
        G, pos,
        with_labels=True,
        node_color=node_colors,
        node_size=2000,
        font_size=12,
        font_weight="bold",
        edge_color="black"
    )
    plt.title(title)
    plt.show()

# ------------------ Backtracking with MRV + LCV ------------------

# Minimum Remaining Values (MRV) heuristic
def select_unassigned_var(assignment):
    unassigned = [v for v in variables if v not in assignment]
    return min(unassigned, key=lambda var: sum(is_valid(assignment, var, c) for c in colors))


# Least Constraining Value (LCV) heuristic
def order_domain_values(var, assignment):
    def conflicts(val):
        return sum(1 for n in neighbors[var] if n not in assignment and not is_valid({**assignment, var: val}, n, val))
    return sorted(colors, key=conflicts)


heuristic_steps = 0
def backtrack_heuristic(assignment):
    global heuristic_steps
    heuristic_steps += 1

    if len(assignment) == len(variables):
        return assignment

    var = select_unassigned_var(assignment)

    for value in order_domain_values(var, assignment):
```

```
    for value in order_domain_values(var, assignment):
        if is_valid(assignment, var, value):
            assignment[var] = value
            result = backtrack_heuristic(assignment)
            if result:
                return result
            del assignment[var]  # backtrack

    return None


if __name__ == "__main__":
# Backtracking with MRV + LCV
    heuristic_solution = backtrack_heuristic({})
    print("Backtracking with MRV + LCV Solution:", heuristic_solution)
    print("Backtracking with MRV + LCV Steps:", heuristic_steps)
    visualize_coloring(heuristic_solution, f"MRV + LCV Heuristic ({heuristic_steps} steps)")
```

```
Backtracking with MRV + LCV Solution: {'V': 'Red', 'SA': 'Green', 'NSW': 'Blue', 'Q': 'Red', 'NT': 'Blue', 'WA': 'Red', 'T': 'Red'}
Backtracking with MRV + LCV Steps: 8
```