SOURCE CODE

Coloring graph

```
Colors = ['red', 'blue', 'green', 'Yellow', 'black']
States = ['andhra', 'Karnataka', 'tamilnadu', 'Kerala']
neighbors = {}
neighbors ['andhra'] = ['Karnataka', 'tamilnadu']
neighbors ['Karnataka'] = ['andhra', 'tamilnadu', 'Kerala']
neighbors ['tamilnadu'] = ['andhra', 'Karnataka', 'Kerala']
neighbors ['Kerala'] = ['Karnataka', 'tamilnadu']

Colors_of_states = {}
def promising(state, Color):
    for neighbor in neighbors.get(state):
        Color_of_neighbor = Colors_of_states.get
                                        (neighbor)
        if Color_of_neighbor == Color:
            return False
    return True
def get_Color_for_state(state):
    for Color in Colors:
        if promising(state, Color):
            return Color

def main():
    for state in states:
        Colors_of_states[state] = get_Color_for
                                    state(state)
    print(Colors_of_states)
main()
```

Travelling Salesman problem:-

```
from sys import maxsize
from itertools import permutations
V = 4
def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation = permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
    return min_path
if __name__ == "__main__":
    graph = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30],
             [20, 25, 30, 0]]
    s = 0
    Print(travellingsalesmanProblem(graph, s))
```

OUTPUT

Graph Coloring Problem:-

{'Andhra': 'red', 'Karnatkaka': 'blue', 'tamilnadu': 'green', 'Kerala': 'red'}

Travelling Salesman problem:-

80

Result:-

Using IDLE (Python 3.9.64-bit) we have succesfully executed Travelling Salesman & Graph Coloring Problems

VIVA QUESTIONS

1. How will you identify a cycle in an undirected graph?

Ans. To detect a cycle in an undirected graph, perform DFS and check if you revisit a node that is not the parent of the current node. If such a node is found, a cycle exists.

2. List out the applications of travelling salesman problem.

Ans. logistics, transportation planning, DNA sequencing, manufacture of microchips.

3. What is graph coloring problem?

Ans. Assigning Colors to specific elements of a graph while adhering to particular limits and constraints.

4. What are the minimum steps to color the tree with given color?

Ans. To Color a tree, use a greedy algorithm to assign Colors ensuring no two adjacent nodes share the same Color. For a tree, you typically need just two Colors.

5. What is NP-Complete problem?

Ans. If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-Complete.

Week-3

AIM:-

Write a program to Implement Travelling Salesman Problem and Graph Coloring Problem.

Algorithm:-

Travelling Saleman Problem:-

- Consider City 1 as the starting and ending point.
- Generate all (n-1)! Permutations of cities.
- Calculate Cost of every permutation and keep track of minimum Cost permutation.
- Return the permutation with minimum Cost.

Graph Coloring Problem:-

- Color first vertex with first Color.
- DO following for remaining V-1 vertices.

a) Consider the Currently picked vertex and Color it with the lowest numbered Color that has not been used on any previously Colored vertices adjacent to it. If all previously used Colors appear on vertices adjacent to v, assign a new Color to it.

Source Code:

Travelling Saleman Problem:-

```
from sys import maxsize
from itertools import permutations
V = 4
def travellingSalesmanProblem(graph, s):
    vertex = []
    for i in range(V):
```

```
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation = permutations(vertex)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
    return min_path
if __name__ == "__main__":
    graph = [[0,10,15,20], [10,0,35,25],
             [15,35,0,30], [20,25,30,0]]
    s = 0
    Print(travellingSalemanProblem(graph,s))
```

Graph Coloring Problem:-

```
colors = ['red', 'blue', 'green', 'yellow', 'black']
states = ['andhra', 'Karnataka', 'tamilnadu', 'kerala']
neighbors = {}
neighbors['andhra'] = ['karnataka', 'tamilnadu']
neighbors['karnataka'] = ['andhra', 'tamilnadu', 'kerala']
neighbors['tamilnadu'] = ['andhra', 'karnataka', 'kerala']
```

```
neighbors['Kerala'] = ['Karnataka','tamilnadu']
Colors_of_states = {}
def promising(state, Color):
    for neighbor in neighbors.get(state):
        Color_of_neighbor = Colors_of_states.get(neighbor)
        if Color_of_neighbor == Color:
            return False
    return True
def get_Color_for_state(state):
    for color in Colors:
        if promising(state, Color):
            return Color

def main():
    for state in states:
        Colors_of_states[state] = get_Color_for state
                                  (state)

    Print(Colors_of_states)
main()
```

Output:-

Travelling Salesman Problem:-

80

Graph Coloring Problem:-

{'andhra':'red','Karnataka':'blue','tamilnadu':'green',
'Kerala':'red'}

Result:

Using IDLE (Python 3.9.64-bit) we have succesfully executed Travelling Salesman & Graph Coloring Problems