SOURCE CODE Coloring graph

Colors = ['red'], blue', 'green', 'Yellow', black'] States = ['andhra', 'Karnataka', 'tamilnadu', 'Kerala'] neighbors = { } neighbors . ['andhra'] = ['Karnataka', 'tarnilnadu'] neighbors ['Karnataka'] = ['andhra', 'tamilnadu', 'Kenala'] neighbors ['tamilnadu'] : ['and hva', 'Kernataka'], 'Kerala'] neighbors [Kevala"] = [Karnataka', 'tamilnadu'] Colors_of_states = { 3 del 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: ed colors _ of _ states [state] = get_ Color_for State (state) print (Colors - of - states)

```
Travelling Salerman problem:
from sys import max 8:3e
from itertools import permutations
def travelling Salesman Problem (graph, s):
     Vertex = []
     for i in range (V):
          if i! = S:
             Vertex.append (i)
     min-path = max size
     next permutation = permutations (vertex)
    for i in next-permutation:
         Current-pathweight = 0
          K = S
         for i in i:
             Current-pathweight += graph[K][j]
         Corrent_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]
   8 = 0
   Print (travellingsalerman Problem (graph,s))
```

OUTPUT

Graph Coloring Problem: f'andhra': 'red', 'Karnatkaka': 'blue', 'tamilnadu': green, Kevala': 'red'3 Travelling Salerman problem: the logistics, transportation planning, DNA Sea manufacture of microchips. -Assigning Clars to specific alaments of a group while adhering to particular limits and constraint. en To Glora tree, on a greety along the do and order enzoone no two radianet meder share the sine Color. For a tree, you typically roced just two colors. mozansldag 9/4 sollo la han 9/1 ai mollog a fit m rolynomial time reducible do id, the problem is NP Complete Using 10te (Python 3.9.64-bit) we have successfully executed Travelling Solerman & Graph Coloring Problems

VIVA QUESTIONS

- 1. How will you identify a cycle in an undirected graph? deed private of the priv
- Ans To detect a cycle in an undirected graph, perform DES and check if you reviset a node that is not the parent of the Correct node. It such a node is found, a cycle exists.
- List out the applications of travelling salesman problem.
- Ans. logistics, transportation planning, DNA sequencing. manufacture of microchips.
- What is graph coloring problem?
- Ans. Assigning Colors to specific elements of a graph while adhering to particular limits and constraints.
- What are the minimum steps to color the tree with given color?
- Ans. To Colora tree, use a greedy algorithm to assign Colors ensuring no two adjacent nocles share the same Color. For a free, you typically need just two Colors.
- 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.

Ostna tole (Fython 3.9. 64-61) we have socienfully enecoded Travelling Salaman & Graph Colored Froblems

AIM:- (1) brogge water Write a program to Implement Travelling Salerman Problem and Evraph Coloring Problem.

Algorithms: i soft along squitered it is ref.

Travelling Saleman Problem: From they home

- · Consider City I 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.

Graphy Coloring Problem: The alma they will

- · Color first vertex with first Color.
- · Do following for remaining V-1 vertices.
- a) Consider the Corrently Picked Verten 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 moldor annished paille work) has

Source Code:
Travelling Saleman Problem:

from sys, import maxsize from itertools imports permutations

V=4

def • travelling Sales man Problem (graph, s):

Vertex = [] for i in range (v):

```
it i != s: 8 40014
                Vertex. append (i)
    min-path = max size

next-permutation = permutations (Verten)
       for i in next-permutation:
            arrent-pathweight =0
  In and could be so producted out to I who when a
         for join is to home ! (1-1) the storense.
     Current_pathweight += graph(K) [j].
                apis from sy tens aveniain & Hourt
    Current_pathweight += gnaph[k][s]
            min-porth = min (min-parth, Current-parthweight)
     return min-path with alla xotist toxit volo) .
  -- name = - == "- main" :

graph = [[0,10,15,20], [10,0,35,25],

[15,35,0,30], [20,25,30,0]]

S = 0

Print (fravelling Sale man Problem (graph, S))
Graph Coloring Problem:-
colors = ['red', 'blue', 'green', 'yellow', 'black']
States = ['andhra', 'Karnataka', 'tamihadu', 'kerala']
neighbors = 93
neighbors { and hra'] = [ Karnataka', Hamilnadu']
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: Veturn False return True def get_color_for_state (state): for Cotor in Colors: if promising (state, color): return Color def main(): for state in states: Colors - of _ states [state] = get_ Color-for state Print (Colors of states) main () Output: Travelling Salesman Problem: Graph Coloring Problem: {'andhra': 'red', 'karnataka': blue', 'tamilnadu': green', 'kerala': 'red'g Resultion 3 9 9 14 1 Using IDLE (Python 3.9.64-bit) we have succenfully exented Travelling Salesman & Graph Coloring Problems