Name: Sujal Singh

Sec: A4_B4_52

Batch: B4

PRACTICAL NO. 7

Aim: Implement Hamiltonian Cycle using Backtracking.
Problem Statement:
The Smart City Transportation Department is designing a night-patrol route for security vehicles.
Each area of the city is represented as a vertex in a graph, and a road between two
areas is represented as an edge.
The goal is to find a route that starts from the main headquarters (Area A), visits each area exactly once, and returns back to the headquarters — forming a Hamiltonian Cycle.
If such a route is not possible, display a suitable message.
1) Adjacency Matrix

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	1
E	1	0	0	1	0

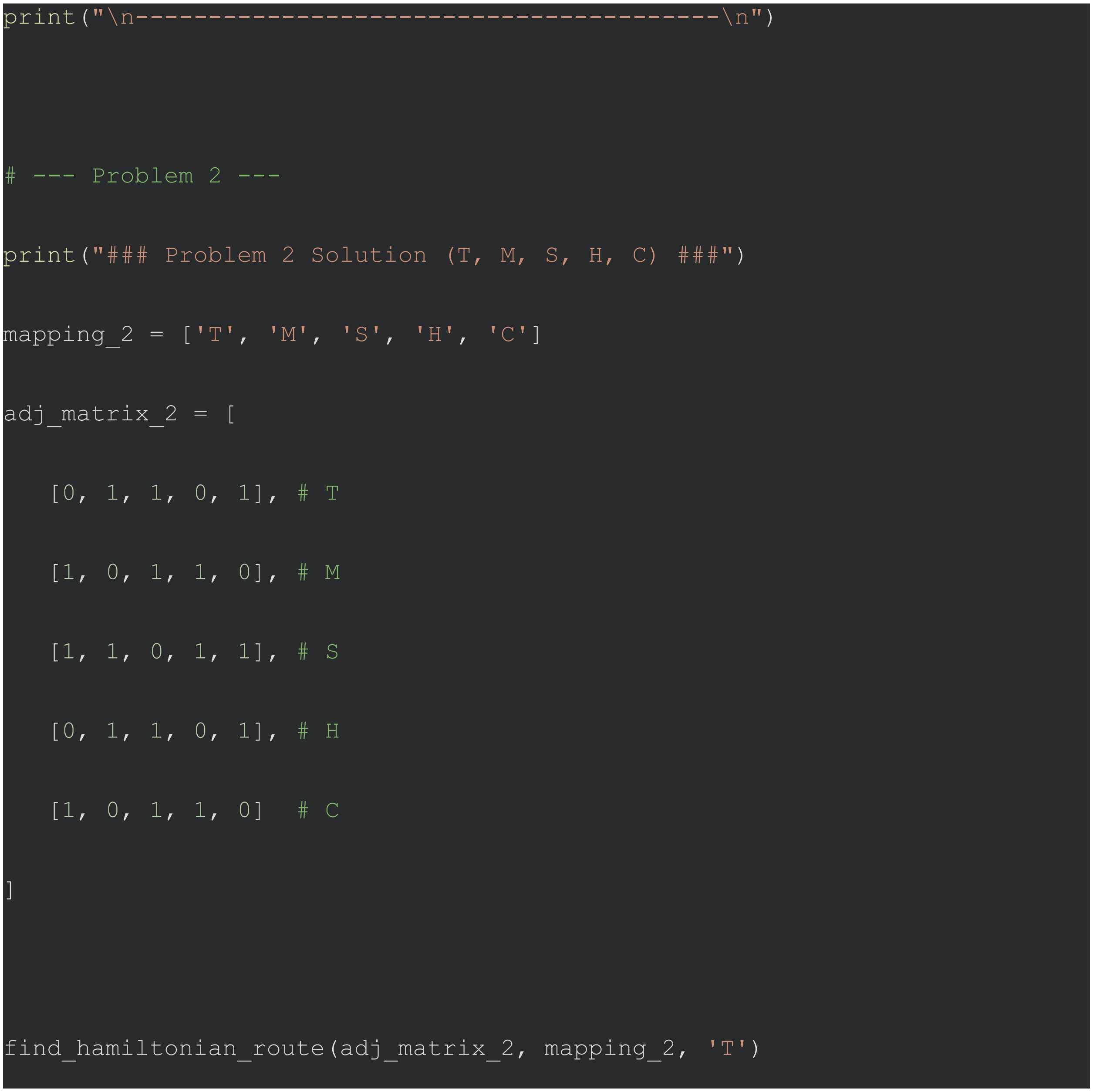
1) Adjacency Matrix

	T	M	S	H	C
T	0	1	1	0	1
M	1	0	1	1	0
S	1	1	0	1	1
H	0	1	1	0	1
C	1	0	1	1	0

Code:

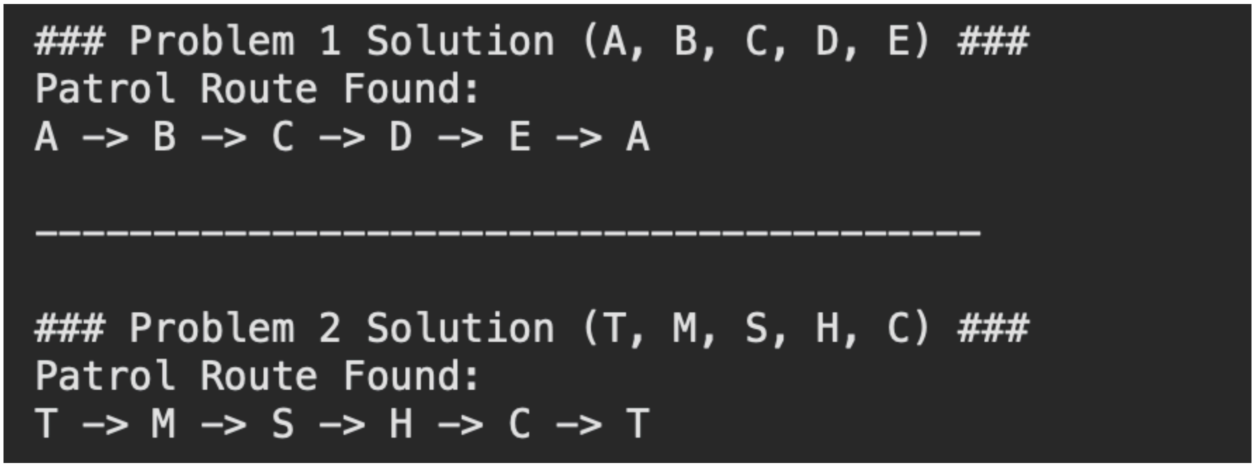
```
def print_solution(path, mapping):
   print("Patrol Route Found:")
   route_str = []
   for vertex_index in path:
       route_str.append(mapping[vertex_index])
   print(" -> ".join(route_str))


def is_safe(v, graph, path, visited):
   last_vertex_in_path = path[-1]


   if graph[last_vertex_in_path][v] == 0:
       return False


   if visited[v] == True:
       return False


   return True


def find_cycle_recursive(graph, mapping, path, visited, V):
```

```python
if len(path) == V:
    last_vertex = path[-1]
    start_vertex = path[0]

    if graph[last_vertex][start_vertex] == 1:
        path.append(start_vertex)
        print_solution(path, mapping)
        return True
    else:
        return False

for v in range(V):
    if is_safe(v, graph, path, visited):

        path.append(v)
        visited[v] = True

        if find_cycle_recursive(graph, mapping, path, visited, V):
                return True

            visited[v] = False
            path.pop()

    return False

def find_hamiltonian_route(graph, mapping, start_area_name):
    V = len(graph)
    path = []
    visited = [False] * V

    try:
        start_index = mapping.index(start_area_name)
    except ValueError:
        print(f"Error: Start area '{start_area_name}' not found.")
        return

    path.append(start_index)
    visited[start_index] = True


    if find_cycle_recursive(graph, mapping, path, visited, V) == False:
        print(f"No Hamiltonian Cycle possible starting from
 {start_area_name}.")




 # --- Problem 1 ---
 print("### Problem 1 Solution (A, B, C, D, E) ###")
 mapping_1 = ['A', 'B', 'C', 'D', 'E']
 adj_matrix_1 = [
    [0, 1, 1, 0, 1], # A
    [1, 0, 1, 1, 0], # B
    [1, 1, 0, 1, 0], # C
    [0, 1, 1, 0, 1], # D
    [1, 0, 0, 1, 0]  # E
 ]


 find_hamiltonian_route(adj_matrix_1, mapping_1, 'A')
```



```python
print("\n----------------------------------------\n")



# --- Problem 2 ---

print("### Problem 2 Solution (T, M, S, H, C) ###")

mapping_2 = ['T', 'M', 'S', 'H', 'C']

adj_matrix_2 = [

   [0, 1, 1, 0, 1], # T

   [1, 0, 1, 1, 0], # M

   [1, 1, 0, 1, 1], # S

   [0, 1, 1, 0, 1], # H

   [1, 0, 1, 1, 0]  # C

]



find_hamiltonian_route(adj_matrix_2, mapping_2, 'T')
```

Output:



```
### Problem 1 Solution (A, B, C, D, E) ###
Patrol Route Found:
A -> B -> C -> D -> E -> A

----------------------------------------

### Problem 2 Solution (T, M, S, H, C) ###
Patrol Route Found:
T -> M -> S -> H -> C -> T
```