Artificial Intelligence Assignment 2

By Himanshu Sardana

Question 1

Write python code for the 8 puzzle problem by taking the following initial and final states

Initial State

```
1 2 3
8   4
7 6 5
```

Final State

```
2 8 1
  4 3
7 6 5
```

Solution:

```python
import copy

initial = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
final = [[2, 8, 1], [0, 4, 3], [7, 6, 5]]

def find_zero(matrix):
    for i in range(3):
        for j in range(3):
            if matrix[i][j] == 0:
                return i, j

def move_up(matrix):
    i, j = find_zero(matrix)
    if i == 0:
        return None
    else:
        new_matrix = copy.deepcopy(matrix)
        new_matrix[i][j], new_matrix[i-1][j] = new_matrix[i-1][j], new_matrix[i][j]
        return new_matrix

def move_down(matrix):
    i, j = find_zero(matrix)
    if i == 2:
        return None
    else:
        new_matrix = copy.deepcopy(matrix)
        new_matrix[i][j], new_matrix[i+1][j] = new_matrix[i+1][j], new_matrix[i][j]
        return new_matrix

def move_left(matrix):
```

```
    i, j = find_zero(matrix)
    if j == 0:
        return None
    else:
        new_matrix = copy.deepcopy(matrix)
        new_matrix[i][j], new_matrix[i][j-1] = new_matrix[i][j-1], new_matrix[i][j]
        return new_matrix

def move_right(matrix):
    i, j = find_zero(matrix)
    if j == 2:
        return None
    else:
        new_matrix = copy.deepcopy(matrix)
        new_matrix[i][j], new_matrix[i][j+1] = new_matrix[i][j+1], new_matrix[i][j]
        return new_matrix

def is_goal(matrix):
    return matrix == final

def bfs(matrix):
    queue = [matrix]
    visited = set()
    visited.add(tuple(map(tuple, matrix)))

    while queue:
        node = queue.pop(0)

        if is_goal(node):
            return node

        for move in [move_up, move_down, move_left, move_right]:
            new_state = move(node)
            if new_state and tuple(map(tuple, new_state)) not in visited:
                visited.add(tuple(map(tuple, new_state)))
                queue.append(new_state)

    return False

def print_matrix(matrix):
    for i in matrix:
        for j in i:
            if j == 0:
                print(' ', end=' ')
            else:
                print(j, end=' ')
        print("")
    print()

result = bfs(initial)

if result:
```

```
    print("Solution found:")
    print_matrix(result)
else:
    print("No solution found.")
```

Question 2

Given 2 jugs of 4 litre and 3 litre capacities. Neither has any measurable markers on it. There is a pump which can be used to fill the jugs with water. Simulate the procedure in Python to get exactly 2 litre of water in the 4 litre jug.

Solution:

```
x = 0
y = 0

x_capacity = 3
y_capacity = 4

target = 2

visited = set()

queue = [(0, 0)]
parent_map = {}
parent_map[(0, 0)] = None

while queue:
    current = queue.pop(0)
    x, y = current

    if y == target:
        break

    if current in visited:
        continue
    visited.add(current)

    next_states = [
        (x_capacity, y),
        (x, y_capacity),
        (0, y),
        (x, 0),
        (x - min(x, y_capacity - y), y + min(x, y_capacity - y)),
        (x + min(y, x_capacity - x), y - min(y, x_capacity - x))
    ]

    for state in next_states:
        if state not in visited:
            queue.append(state)
            parent_map[state] = current

path = []
```

```
while current:
    path.append(current)
    current = parent_map[current]

if path:
    print("Solution:")
    for step in path[::-1]:
        print(f"Jug 1: {step[0]}L, Jug 2: {step[1]}L")
else:
    print("No solution exists for the given inputs.")
```

Question 3

Write a python program to implement Travelling Salesman Problem (TSP). Take the starting node from the user at runtime

Solution:

```
def travelling_salesman(graph, starting_node):
    visited = set()
    visited.add(starting_node)

    current_node = starting_node
    total_cost = 0
    path = [starting_node]

    while len(visited) < len(graph):
        unvisited_neighbors = {node: cost for node, cost in
graph[current_node].items() if node not in visited}
        if not unvisited_neighbors:
            break
        next_node = min(unvisited_neighbors, key=unvisited_neighbors.get)
        total_cost += graph[current_node][next_node]
        visited.add(next_node)
        path.append(next_node)
        current_node = next_node

    total_cost += graph[current_node][starting_node]
    path.append(starting_node)

    return total_cost, path


graph = {
    1: {2: 10, 3: 15, 4: 20},
    2: {1: 10, 3: 35, 4: 25},
    3: {1: 15, 2: 35, 4: 30},
    4: {1: 20, 2: 25, 3: 30}
}

starting_node = int(input("Enter the starting node: "))

cost, path = travelling_salesman(graph, starting_node)
```

```
print(f"Total cost: {cost}")
print(f"Path taken: {path}")
```