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 heapq
from copy import deepcopy

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

def find_blank(state):
    for i in range(len(state)):
        for j in range(len(state[i])):
            if state[i][j] == 0:
                return [i, j]
    return None

def manhattan_distance(state, goal):
    distance = 0
```

```
    for i in range(len(state)):
        for j in range(len(state[i])):
            if state[i][j] != 0:
                goal_i, goal_j = next((x, y) for x in range(len(goal)) for y in range(len(goal[x])) if goal[x][y] == state[i][j])
                distance += abs(i - goal_i) + abs(j - goal_j)
    return distance

def get_neighbors(state):
    neighbors = []
    blank = find_blank(state)
    x, y = blank[0], blank[1]

    # Directions: [name, dx, dy]
    directions = [
        ("up", -1, 0),
        ("down", 1, 0),
        ("left", 0, -1),
        ("right", 0, 1)
    ]

    for name, dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < len(state) and 0 <= ny < len(state[0]):  # Ensure move is within bounds
            new_state = deepcopy(state)
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append((name, new_state))

    return neighbors

def solve_puzzle(initial, goal):
    pq = []
    heapq.heappush(pq, (0, initial, []))

```

```
    visited = set()

    while pq:
        f_score, current_state, path = heapq.heappop(pq)

        if current_state == goal:
            return path

        state_tuple = tuple(tuple(row) for row in current_state)
        if state_tuple in visited:
            continue
        visited.add(state_tuple)

        for move_name, neighbor in get_neighbors(current_state):
            if tuple(tuple(row) for row in neighbor) not in visited:
                g_score = len(path) + 1  # Cost so far
                h_score = manhattan_distance(neighbor, goal)  # Heuristic
                f_score = g_score + h_score  # Total cost
                heapq.heappush(pq, (f_score, neighbor, path + [move_name]))

    return None  # No solution found

solution = solve_puzzle(initial, final)
print(solution)
```

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:

```
from collections import deque

def water_jug_problem(jug1_capacity, jug2_capacity, target):
    visited = set()

```

```
    queue = deque()
    queue.append((0, 0))

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

    while queue:
        current = queue.popleft()
        jug1, jug2 = current

        if jug2 == target:
            return reconstruct_path(parent_map, current)

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

        next_states = [
            (jug1_capacity, jug2),
            (jug1, jug2_capacity),
            (0, jug2),
            (jug1, 0),
            (jug1 - min(jug1, jug2_capacity - jug2), jug2 + min(jug1,
↪   jug2_capacity - jug2)),
            (jug1 + min(jug2, jug1_capacity - jug1), jug2 - min(jug2,
↪   jug1_capacity - jug1))
        ]

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

    return None

def reconstruct_path(parent_map, state):
```

```
    path = []
    while state:
        path.append(state)
        state = parent_map[state]
    return path[::-1]

jug1_capacity = 3
jug2_capacity = 4
target = 2

result = water_jug_problem(jug1_capacity, jug2_capacity, target)

if result:
    print("Solution:")
    for step in result:
        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}")
```