8-PUZZLE Problem in AI using A*

```
import heapq
goal = (1,2,3,4,5,6,7,8,0)
def manhattan(state):
    dist = 0
    for i in range(9):
        if state[i] != 0:
            goal_pos = goal.index(state[i])
            dist += abs(i//3 - goal_pos//3) + abs(i%3 - goal_pos%3)
    return dist
def get_neighbors(state):
    neighbors = []
    idx = state.index(0)
    moves = { -3, 3, -1, 1 }
    for m in moves:
        new = idx + m
        if 0 <= new < 9:
            new_state = list(state)
            new_state[idx], new_state[new] = new_state[new], new_state[idx]
            print(new_state)
            neighbors.append(tuple(new_state))
    return neighbors
def astar(start):
    pq = [(manhattan(start), 0, start)]
    visited = set()
    while pq:
        f, g, state = heapq.heappop(pq)
        if state == goal:
            print("Solved!")
            return
        if state in visited:
            continue
        visited.add(state)
        for n in get_neighbors(state):
            heapq.heappush(pq, (g + 1 + manhattan(n), g + 1, n))
start = (1,2,3,4,0,5,6,7,8)
astar(start)
```

WATER in JUGS Problem in AI using BFS

```
from collections import deque
def water_jug():
    start = (0, 0)
    goal = 2
    visited = set()
    q = deque([(start, [start])])  # store (state, path)

    while q:
        (x, y), path = q.popleft()

        # Check goal
        if x == goal or y == goal:
            print("Goal reached:", (x, y))
            print("Action sequence (states):")
            for step in path:
                print(step)
            return

        if (x, y) in visited:
            continue
        visited.add((x, y))

        # Generate possible moves
        actions = [
            (4, y),          # Fill A
            (x, 3),          # Fill B
            (0, y),          # Empty A
            (x, 0),          # Empty B
            (x - min(x, 3-y), y + min(x, 3-y)),  # Pour A→B
            (x + min(y, 4-x), y - min(y, 4-x))   # Pour B→A
        ]

        # Add new states to queue with updated path
        for a in actions:
            if a not in visited:
                q.append((a, path + [a]))

water_jug()
```

TRAVELLING SALES PERSON (TSP) Problem in AI using Hill Climbing

```
import random

import math

cities = [(0,0),(1,3),(4,3),(6,1)]

def distance(a, b):

    return math.dist(a, b)

def total_cost(route):

    cost = 0

    for i in range(len(route)):

        cost += distance(route[i], route[(i+1)%len(route)])

    return cost

def hill_climbing():

    current = cities[:]

    random.shuffle(current)

    while True:

        neighbors = []

        for i in range(len(current)):

            for j in range(i+1, len(current)):

                n = current[:]

                n[i], n[j] = n[j], n[i]

                neighbors.append(n)

        best = min(neighbors, key=total_cost)

        if total_cost(best) >= total_cost(current):

            print("Best route:", current)

            print("Cost:", total_cost(current))

            return

        current = best

hill_climbing()
```