exp2.informed search A*

```
import numpy as np
import heapq
class Node:
def __init__(self, state, parent=None, move=None, cost=0, heuristic=0):
self.state = state
self.parent = parent
self.move = move
self.cost = cost
self.heuristic = heuristic
self.total_cost = cost + heuristic
def __lt__(self, other):
return self.total_cost < other.total_cost
def heuristic(state):
"""Calculate the Manhattan distance heuristic for the state."""
distance = 0
goal_state = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
for i in range(3):
for j in range(3):
if state[i][j] != 0:
goal_position = np.argwhere(goal_state == state[i][j])[0]
distance += abs(i - goal_position[0]) + abs(j - goal_position[1])
return distance
def get_neighbors(state):
"""Return all possible neighbors (states) for the given state."""
neighbors = []
zero_position = np.argwhere(state == 0)[0]
x, y = zero_position
moves = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
for new_x, new_y in moves:
if 0 <= new_x < 3 and 0 <= new_y < 3:
```

```
new_state = state.copy()
new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y] # Swap
neighbors.append(new_state)
return neighbors
def a_star(start_state):
"""Perform the A* search algorithm to solve the 8-puzzle problem."""
start_node = Node(start_state, None, None, 0, heuristic(start_state))
open_set = []
closed_set = set()
heapq.heappush(open_set, start_node)
while open_set:
current_node = heapq.heappop(open_set)
if np.array_equal(current_node.state, np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])):
return current_node
closed_set.add(tuple(map(tuple, current_node.state)))
for neighbor in get_neighbors(current_node.state):
neighbor_tuple = tuple(map(tuple, neighbor))
if neighbor_tuple in closed_set:
continue
g_cost = current_node.cost + 1
h_cost = heuristic(neighbor)
neighbor_node = Node(neighbor, current_node, None, g_cost, h_cost)
if neighbor_node not in open_set:
heapq.heappush(open_set, neighbor_node)
return None
def print_solution(solution_node):
"""Print the solution path from start to goal."""
path = []
while solution_node:
path.append(solution_node.state)
solution_node = solution_node.parent
for state in reversed(path):
```

```
print(state)
initial_state = np.array([[1, 2, 3],
[4, 0, 5],
[7, 8, 6]])
solution = a_star(initial_state)
if solution:
print("Solution found:")
print_solution(solution)
else:   print("No solution found.")
```