assign5

January 31, 2025

[3]:
```
import heapq

def misplaced_tiles(state, goal):
    count = 0
    for i in range(9):
        if state[i] != 0 and state[i] != goal[i]:
            count += 1
    return count

def manhattan_distance(state, goal):
    distance = 0
    for i in range(9):
        if state[i] != 0:
            goal_index = goal.index(state[i])
            distance += abs(i // 3 - goal_index // 3) + abs(i % 3 - goal_index␣
 ↪% 3)
    return distance

def get_neighbors(state):
    neighbors = []
    blank_index = state.index(0)
    row, col = blank_index // 3, blank_index % 3
    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for dr, dc in moves:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_state = list(state)
            new_blank_index = new_row * 3 + new_col
            new_state[blank_index], new_state[new_blank_index] =␣
 ↪new_state[new_blank_index], new_state[blank_index]
            neighbors.append(tuple(new_state))
    return neighbors

def astar(start, goal, heuristic):
    open_set = [(heuristic(start, goal), 0, start)]  # f_score, g_score, state
    heapq.heapify(open_set)
```

```
    closed_set = set()
    came_from = {}
    nodes_explored = 0

    while open_set:
        f_score, g_score, current = heapq.heappop(open_set)
        nodes_explored += 1

        if current == goal:
            path = reconstruct_path(came_from, current)
            return path, nodes_explored, len(path) - 1

        if current in closed_set:
            continue

        closed_set.add(current)

        for neighbor in get_neighbors(current):
            tentative_g_score = g_score + 1

            if neighbor not in closed_set or tentative_g_score < g_score:
                came_from[neighbor] = current
                h_score = heuristic(neighbor, goal)
                f_score = tentative_g_score + h_score
                heapq.heappush(open_set, (f_score, tentative_g_score, neighbor))

    return None, nodes_explored, -1

def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    return path[::-1]

def get_user_input():
    while True:
        try:
            raw_input = input("Enter the start state (9 numbers,␣
 ↪space-separated, 0 for blank): ")
            start_state_list = [int(x) for x in raw_input.split()]
            if len(start_state_list) != 9 or any(x not in range(9) for x in␣
 ↪start_state_list):
                raise ValueError("Invalid input. Please enter 9 numbers between␣
 ↪0 and 8.")
            start_state = tuple(start_state_list)
```

```
            raw_input = input("Enter the goal state (9 numbers, space-separated, 0 for blank): ")
            goal_state_list = [int(x) for x in raw_input.split()]
            if len(goal_state_list) != 9 or any(x not in range(9) for x in goal_state_list):
                raise ValueError("Invalid input. Please enter 9 numbers between 0 and 8.")
            goal_state = tuple(goal_state_list)

            return start_state, goal_state
        except ValueError as e:
            print(e)

if __name__ == "__main__":
    start_state, goal_state = get_user_input()

    path_h1, nodes_h1, depth_h1 = astar(start_state, goal_state, misplaced_tiles)
    print("\nA* with Misplaced Tiles:")
    if path_h1:
        print("Path:", path_h1)
        print("Nodes Explored:", nodes_h1)
        print("Solution Depth:", depth_h1)
    else:
        print("No path found.")

    path_h2, nodes_h2, depth_h2 = astar(start_state, goal_state, manhattan_distance)
    print("\nA* with Manhattan Distance:")
    if path_h2:
        print("Path:", path_h2)
        print("Nodes Explored:", nodes_h2)
        print("Solution Depth:", depth_h2)
    else:
        print("No path found.")
```

```
Enter the start state (9 numbers, space-separated, 0 for blank):  3 1 2 4 5 6 7 8 9 5

Invalid input. Please enter 9 numbers between 0 and 8.

Enter the start state (9 numbers, space-separated, 0 for blank):  1 2 3 4 8 6 7 5 0
Enter the goal state (9 numbers, space-separated, 0 for blank):  1 2 3 4 5 6 7 8 9

Invalid input. Please enter 9 numbers between 0 and 8.

Enter the start state (9 numbers, space-separated, 0 for blank):  1 2 3 4 5 8 6
```

```
7 0
Enter the goal state (9 numbers, space-separated, 0 for blank):  1 2 3 4 5 6 7 0
8


A* with Misplaced Tiles:
Path: [(1, 2, 3, 4, 5, 8, 6, 7, 0), (1, 2, 3, 4, 5, 8, 6, 0, 7), (1, 2, 3, 4, 5,
8, 0, 6, 7), (1, 2, 3, 0, 5, 8, 4, 6, 7), (1, 2, 3, 5, 0, 8, 4, 6, 7), (1, 2, 3,
5, 6, 8, 4, 0, 7), (1, 2, 3, 5, 6, 8, 4, 7, 0), (1, 2, 3, 5, 6, 0, 4, 7, 8), (1,
2, 3, 5, 0, 6, 4, 7, 8), (1, 2, 3, 0, 5, 6, 4, 7, 8), (1, 2, 3, 4, 5, 6, 0, 7,
8), (1, 2, 3, 4, 5, 6, 7, 0, 8)]
Nodes Explored: 79
Solution Depth: 11

A* with Manhattan Distance:
Path: [(1, 2, 3, 4, 5, 8, 6, 7, 0), (1, 2, 3, 4, 5, 8, 6, 0, 7), (1, 2, 3, 4, 5,
8, 0, 6, 7), (1, 2, 3, 0, 5, 8, 4, 6, 7), (1, 2, 3, 5, 0, 8, 4, 6, 7), (1, 2, 3,
5, 6, 8, 4, 0, 7), (1, 2, 3, 5, 6, 8, 4, 7, 0), (1, 2, 3, 5, 6, 0, 4, 7, 8), (1,
2, 3, 5, 0, 6, 4, 7, 8), (1, 2, 3, 0, 5, 6, 4, 7, 8), (1, 2, 3, 4, 5, 6, 0, 7,
8), (1, 2, 3, 4, 5, 6, 7, 0, 8)]
Nodes Explored: 42
Solution Depth: 11
```

[9]:
```python
import heapq

# ... (misplaced_tiles, manhattan_distance, get_neighbors,
 #     astar, reconstruct_path, get_user_input functions - same as before)


def run_test(start_state, goal_state):
    print(f"\n--- Testing with Start State: {start_state} ---")

    path_h1, nodes_h1, depth_h1 = astar(start_state, goal_state,␣
 ↪misplaced_tiles)
    print("\nA* with Misplaced Tiles:")
    if path_h1:
        print("Nodes Explored:", nodes_h1)
        print("Solution Depth:", depth_h1)
    else:
        print("No path found.")

    path_h2, nodes_h2, depth_h2 = astar(start_state, goal_state,␣
 ↪manhattan_distance)
    print("\nA* with Manhattan Distance:")
    if path_h2:
        print("Nodes Explored:", nodes_h2)
        print("Solution Depth:", depth_h2)
    else:
```

```
        print("No path found.")


if __name__ == "__main__":
    goal_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)  # Define goal state ONCE

    test_states = [
        (1, 2, 3, 4, 8, 6, 7, 5, 0),  # Test 1 (Moderate difficulty)
        (1, 2, 3, 5, 8, 6, 4, 7, 0),  # Test 2 (Slightly harder)
        (4, 1, 3, 2, 8, 6, 7, 5, 0),  # Test 3 (Rearranged tiles)
        (8, 1, 2, 4, 5, 3, 7, 6, 0),  # Test 4 (More scrambled)
        (8, 6, 7, 2, 5, 4, 3, 1, 0),  # Test 5 (Harder - Inversions present)
    ]

    for start_state in test_states:
        run_test(start_state, goal_state)


    # You can still get user input if you want:
   #  start_state, goal_state = get_user_input()  # Uncomment if needed
    # run_test(start_state, goal_state)       # Uncomment if needed
```

```
--- Testing with Start State: (1, 2, 3, 4, 8, 6, 7, 5, 0) ---

A* with Misplaced Tiles:
No path found.

A* with Manhattan Distance:
No path found.

--- Testing with Start State: (1, 2, 3, 5, 8, 6, 4, 7, 0) ---

A* with Misplaced Tiles:
No path found.

A* with Manhattan Distance:
No path found.

--- Testing with Start State: (4, 1, 3, 2, 8, 6, 7, 5, 0) ---

A* with Misplaced Tiles:
No path found.

A* with Manhattan Distance:
No path found.
```

```
--- Testing with Start State: (8, 1, 2, 4, 5, 3, 7, 6, 0) ---

A* with Misplaced Tiles:
Nodes Explored: 248
Solution Depth: 14

A* with Manhattan Distance:
Nodes Explored: 75
Solution Depth: 14

--- Testing with Start State: (8, 6, 7, 2, 5, 4, 3, 1, 0) ---

A* with Misplaced Tiles:
Nodes Explored: 141253
Solution Depth: 30

A* with Manhattan Distance:
Nodes Explored: 12574
Solution Depth: 30
```

[ ]: