**Week 2**

Misplaced Tiles – 8 puzzle

Code:

import heapq

# ---------- Heuristic: Misplaced Tiles ----------

def misplaced\_tiles(state, goal):

    """Count number of misplaced tiles (ignores blank 0)."""

    return sum(1 for i in range(len(state)) if state[i] != 0 and state[i] != goal[i])

# ---------- Puzzle Neighbors ----------

def get\_neighbors(state):

    neighbors = []

    idx = state.index(0) # position of blank

    x, y = divmod(idx, 3)

    moves = [(-1,0),(1,0),(0,-1),(0,1)] # up, down, left, right

    for dx, dy in moves:

        nx, ny = x + dx, y + dy

        if 0 <= nx < 3 and 0 <= ny < 3:

            new\_idx = nx \* 3 + ny

            new\_state = list(state)

            new\_state[idx], new\_state[new\_idx] = new\_state[new\_idx], new\_state[idx]

            neighbors.append(tuple(new\_state))

    return neighbors

# ---------- A\* Search ----------

def a\_star\_misplaced(start, goal):

    open\_list = []

    heapq.heappush(open\_list, (misplaced\_tiles(start, goal), 0, start, [start]))

    closed = set()

    while open\_list:

        f, g, state, path = heapq.heappop(open\_list)

        if state == goal:

            return path # solution found

        if state in closed:

            continue

        closed.add(state)

        for neighbor in get\_neighbors(state):

            if neighbor not in closed:

                g\_new = g + 1

                h\_new = misplaced\_tiles(neighbor, goal)

                f\_new = g\_new + h\_new

                heapq.heappush(open\_list, (f\_new, g\_new, neighbor, path + [neighbor]))

    return None # no solution found

# ---------- Run Example ----------

if \_\_name\_\_ == "\_\_main\_\_":

    start = (1, 2, 3,

             4, 5, 6,

             0, 7, 8)

    goal = (1, 2, 3,

            4, 5, 6,

            7, 8, 0)

    solution = a\_star\_misplaced(start, goal)

    if solution:

        print("Solution found in", len(solution)-1, "moves.")

        for step in solution:

            for i in range(0, 9, 3):

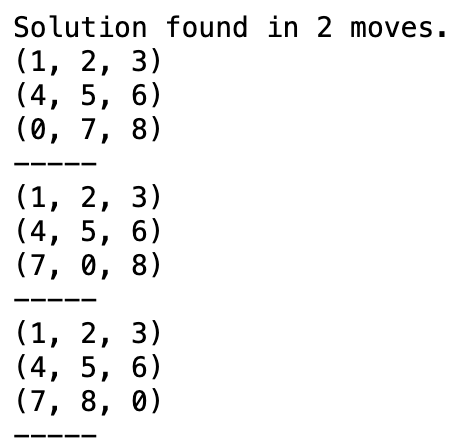
                print(step[i:i+3])

            print("-----")

    else:

        print("No solution found.")

Output:



Manhattan distance – 8 puzzle

import heapq

# ---------- Heuristic: Manhattan Distance ----------

def manhattan\_distance(state, goal):

    """Sum of Manhattan distances of each tile from its goal position."""

    distance = 0

    for i, tile in enumerate(state):

        if tile != 0: # skip the blank

            goal\_pos = goal.index(tile)

            distance += abs(i // 3 - goal\_pos // 3) + abs(i % 3 - goal\_pos % 3)

    return distance

# ---------- Puzzle Neighbors ----------

def get\_neighbors(state):

    neighbors = []

    idx = state.index(0) # blank position

    x, y = divmod(idx, 3)

    moves = [(-1,0),(1,0),(0,-1),(0,1)] # up, down, left, right

    for dx, dy in moves:

        nx, ny = x + dx, y + dy

        if 0 <= nx < 3 and 0 <= ny < 3:

            new\_idx = nx \* 3 + ny

            new\_state = list(state)

            new\_state[idx], new\_state[new\_idx] = new\_state[new\_idx], new\_state[idx]

            neighbors.append(tuple(new\_state))

    return neighbors

# ---------- A\* Search ----------

def a\_star\_manhattan(start, goal):

    open\_list = []

    heapq.heappush(open\_list, (manhattan\_distance(start, goal), 0, start, [start]))

    closed = set()

    while open\_list:

        f, g, state, path = heapq.heappop(open\_list)

        if state == goal:

            return path # solution found

        if state in closed:

            continue

        closed.add(state)

        for neighbor in get\_neighbors(state):

            if neighbor not in closed:

                g\_new = g + 1

                h\_new = manhattan\_distance(neighbor, goal)

                f\_new = g\_new + h\_new

                heapq.heappush(open\_list, (f\_new, g\_new, neighbor, path + [neighbor]))

    return None # no solution found

# ---------- Run Example ----------

if \_\_name\_\_ == "\_\_main\_\_":

    start = (1, 2, 3,

             4, 5, 6,

             0, 7, 8)

    goal = (1, 2, 3,

            4, 5, 6,

            7, 8, 0)

    solution = a\_star\_manhattan(start, goal)

    if solution:

        print("Solution found in", len(solution)-1, "moves.")

        for step in solution:

            for i in range(0, 9, 3):

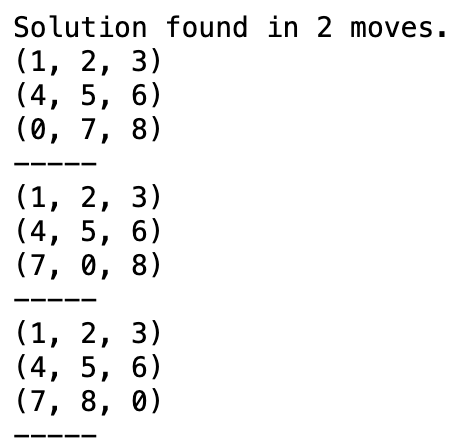
                print(step[i:i+3])

            print("-----")

    else:

        print("No solution found.")

Output:



Iterative Deepening Depth First Search

# ---------- Depth Limited Search ----------

def DLS(graph, node, goal, limit, visited):

    if node == goal:

        return True

    if limit == 0:

        return False

    visited.add(node)

    for neighbor in graph.get(node, []):

        if neighbor not in visited:

            if DLS(graph, neighbor, goal, limit - 1, visited):

                return True

    return False

# ---------- IDDFS ----------

def IDDFS(graph, start, goal, max\_depth):

    for depth in range(max\_depth + 1):

        visited = set()

        if DLS(graph, start, goal, depth, visited):

            return True

    return False

# ---------- Example Run ----------

if \_\_name\_\_ == "\_\_main\_\_":

    graph = {

        'A': ['B', 'C'],

        'B': ['D', 'E'],

        'C': ['F'],

        'D': [],

        'E': ['F'],

        'F': []

    }

    start = 'A'

    goal = 'F'

    if IDDFS(graph, start, goal, max\_depth=3):

        print(f"Goal {goal} found within depth limit.")

    else:

        print(f"Goal {goal} not found within depth limit.")

