Week 3:

A*_ManhattanDistanceA

CODE:

```
#Manhattan approach
import heapq


def solve_8puzzle(initial_state):
    goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
    priority_queue = [(heuristic(initial_state, goal_state), 0, initial_state, [])]
    visited = set()

    while priority_queue:
        f_cost, g_cost, current_state, current_path = heapq.heappop(priority_queue)

        if current_state == goal_state:
            return current_path + [current_state]

        if tuple(map(tuple, current_state)) in visited:
            continue
        visited.add(tuple(map(tuple, current_state)))

        for next_state, action in get_possible_moves(current_state):
            new_g_cost = g_cost + 1
            new_f_cost = new_g_cost + heuristic(next_state, goal_state)
            heapq.heappush(priority_queue, (new_f_cost, new_g_cost, next_state,
current_path + [(current_state, action)]))

    return None


def heuristic(state, goal_state):
    distance = 0
    for i in range(3):
```

```
        for j in range(3):
            if state[i][j] != 0:
                goal_row, goal_col = find_position(goal_state, state[i][j])
                distance += abs(i - goal_row) + abs(j - goal_col)
    return distance


def find_position(state, tile):
    for i in range(3):
        for j in range(3):
            if state[i][j] == tile:
                return i, j


def get_possible_moves(state):
    row, col = find_position(state, 0)
    possible_moves = []

    if row > 0:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row - 1][col] = new_state[row - 1][col],
new_state[row][col]
        possible_moves.append((new_state, 'Up'))
    if row < 2:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col],
new_state[row][col]
        possible_moves.append((new_state, 'Down'))
    if col > 0:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1],
new_state[row][col]
        possible_moves.append((new_state, 'Left'))
    if col < 2:
        new_state = [list(row) for row in state]
```

```
        new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1],
new_state[row][col]
        possible_moves.append((new_state, 'Right'))

    return possible_moves


initial_state = [[2, 8, 3], [1, 6, 4], [0, 7, 5]]
solution = solve_8puzzle(initial_state)

if solution:
    print("Solution found:")
    for state, action in solution[:-1]:
        print("--------------------")
        for row in state:
            print(row)
        print("Move:", action)
    print("--------------------")
    for row in solution[-1]:
        print(row)
else:
    print("No solution found.")
```

Output:

```
Solution found:
--------------------
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]
Move: Right
--------------------
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
Move: Up
--------------------
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
Move: Up
--------------------
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
Move: Left
--------------------
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
Move: Down
--------------------
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
Move: Right
--------------------
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

(ii) Manhattan Distance

$g(n)$ = depth of the node

$h(n)$ = Manhattan Distance.

2	8	3
1	6	4
7		5

$f(n) = 0 + 5 = 5$

$1+2+1+1$ -

2	8	3
1		4
7	6	5

$f(n) = 1 + 4 = 5$

$1 + 2 + 1 +$

2	8	3
1	6	4
	7	5

$f(n) = 1 + 6 = 7$

2	8	3
1	6	4
7	5	

$f(n) = 1 + 6 = 7$

2		3
1	8	4
7	6	5

$f(n) = 2 + 3 = 5$

2	8	3
	1	4
7	6	5

$f(n) = 2 + 5 = 7$

2	8	3
1	4	
7	6	5

$f(n) = 2 + 5 = 7$

	2	3
1	8	4
7	6	5

$f(n) = 3 + 2 = 5$

2	3	
1	8	4
7	6	5

$f(n) = 3 + 4 = 7$

1	2	3
	8	4
7	6	5

$f(n) = 4 + 1 = 5$

2	·	3
1	8	4
7	6	5

$f(n) = 4 + 2 = 6$.

GOAL STATE

1	2	3
8		4
7	6	5

$f(n) = 5 + 0 = 5$

classmate
Date
Page

Algorithm:

1. Initialize
   - start with initial state of the puzzle
   - Set the goal state

2. Priority queue:
   - use priority queue to store the states of the puzzle prioritized by $f(n) = g(n) + h(n)$
   $g(n) = \text{depth}$
   $h(n) = \text{manhattan distance}$

3. Explore states:
   - Remove the state with the smallest $f(n)$ from the queue
   - If the current state is goal state stop and return
   - Generate all possible new states.

4. Evaluate New states
   & calculate $g(n)$, $h(n)$ $f(n) = g(n) + h(n)$

Repeat ~~the~~
exploring states from the queue until the goal state is reached
Once the goal state is reached algorithm terminates

15/10/24