Assignment 3 (a)

- Given an initial state of an 8-puzzle problem and final state to be reached:-

Initial State

Final State

Find the most cost-effective path to reach the final state from the initial state using the A* Algorithm. Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

Code:

```
 from queue import PriorityQueue

from copy import deepcopy

def a_star(initial_state, goal_state, heuristic):

    frontier = PriorityQueue()

    frontier.put((0, initial_state))

    came_from = {str(initial_state): None}

    cost_so_far = {str(initial_state): 0}
    while not frontier.empty():

        current = frontier.get()[1]



        if current == goal_state:

            break



        for next_state, cost in generate_next_states(initial_state,current):

            new_cost = cost_so_far[str(current)] + cost
```

```
            if str(next_state) not in cost_so_far or new_cost < 
cost_so_far[str(next_state)]:

                cost_so_far[str(next_state)] = new_cost

                priority = new_cost + heuristic(next_state, goal_state)

                frontier.put((priority, next_state))

                came_from[str(next_state)] = current

    path = []

    current = goal_state

    while current != initial_state:

        path.append(current)

        current = came_from[str(current)]

    path.append(initial_state)

    path.reverse()

    return path

def generate_next_states(initial_state,current_state):

    # TODO: Implement function to generate possible next states from a given 
state

    result=[]

    x,y = -1,-1

    n = len(current_state)

    for i in range(n):

        for j in range(n):

            if current_state[i][j] == 0:

                x,y=i,j

    try:
```

```
        mat1 = deepcopy(current_state)
        mat1[x-1][y],mat1[x][y] = mat1[x][y],mat1[x-1][y]
        result.append((mat1,heuristic(initial_state,current_state)))
    except:
        pass
    try:
        mat2 = deepcopy(current_state)
        mat2[x][y],mat2[x+1][y] = mat2[x+1][y],mat2[x][y]
        result.append((mat2,heuristic(initial_state,current_state)))
    except:
        pass
    try:
        mat1 = deepcopy(current_state)
        mat1[x][y-1],mat1[x][y] = mat1[x][y],mat1[x][y-1]
        result.append((mat1,heuristic(initial_state,current_state)))
    except:
        pass

    try:
        mat2 = deepcopy(current_state)
        mat2[x][y],mat2[x][y+1] = mat2[x][y+1],mat2[x][y]
        result.append((mat2,heuristic(initial_state,current_state)))
    except:
        pass
    return result
```

```
def heuristic(state, goal_state):

    # TODO: Implement heuristic function to estimate cost of reaching goal
state from a given state

    n = len(state)

    current_coordinates={}

    goal_coordinates={}

    for i in range(n):

        for j in range(n):

            current_coordinates[state[i][j]] = (i,j)

            goal_coordinates[goal_state[i][j]] = (i,j)

    total_cost = 0

    for key in current_coordinates:

        if key == 0 :

            continue

        x1,y1 = current_coordinates[key]

        x2,y2 = goal_coordinates[key]

        total_cost += abs(x2-x1)+abs(y2-y1)

    return total_cost
# Example usage:

initial_state = [[2, 8, 3],

                 [1, 6, 4],

                 [7, 0, 5]]

goal_state = [[1, 2, 3],

              [8, 0, 4],

              [7, 6, 5]]
```

```
path = a_star(initial_state, goal_state, heuristic)
for i,mat in enumerate (path):

    print('State:' ,(i+1))

    for row in mat:

        print('\t',row)
```

Output:

```
State: 1
         [2, 8, 3]
         [1, 6, 4]
         [7, 0, 5]
State: 2
         [2, 8, 3]
         [1, 0, 4]
         [7, 6, 5]
State: 3
         [2, 0, 3]
         [1, 8, 4]
         [7, 6, 5]
State: 4
         [0, 2, 3]
         [1, 8, 4]
         [7, 6, 5]
State: 5
         [1, 2, 3]
         [0, 8, 4]
         [7, 6, 5]
State: 6
         [1, 2, 3]
         [8, 0, 4]
         [7, 6, 5]
```