```python
import numpy as np

puzzle_tile = np.zeros((3,3))
puzzle_tile = np.argwhere(puzzle_tile==0)
goal_state  = np.array([1, 4, 7, 2, 5, 8, 3, 6, 0])
visited_states = []

##########################################################################################################################################

def print_matrix(state):
    counter = 0
    for row in range(0, len(state), 3):
        if counter == 0 :
            print("-------------")
        for element in range(counter, len(state), 3):
            if element <= counter:
                print("|", end=" ")
            print(int(state[element]), "|", end=" ")
        counter = counter +1
        print("\n-------------")

##########################################################################################################################################

def visit_checker(current_state):
    '''
    Checks if current state is in parent node history
    '''

    truth = False

    for i in visited_states:

        check = np.array_equiv(i,current_state)

        if check:

            return check


    return truth

##########################################################################################################################################

class node:

    def __init__(self, state, parent_node, node_id):

        self.state = state
        self.node_id = node_id

        #For child nodes
        visited_states.append(state)

        #Puzzle config
        self.child_nodes = []
        self.zero_ind = np.argwhere(state==0)[0][0]
        self.child_states = []
        self.parent_node = parent_node

        #Right possible location
        right_loc = np.add(puzzle_tile[self.zero_ind],np.array([0,1]))
        if right_loc[1]>2:
            self.right_loc = []
        else:
            self.right_loc = right_loc

        #Left possible location
        left_loc = np.add(puzzle_tile[self.zero_ind],np.array([0,-1]))
        if left_loc[1]<0:
            self.left_loc = []
        else:
            self.left_loc = left_loc

        #Down possible location
        down_loc = np.add(puzzle_tile[self.zero_ind],np.array([1,0]))
        if down_loc[0]>2:
            self.down_loc = []
        else:
            self.down_loc = down_loc

        #Up possible location
        up_loc = np.add(puzzle_tile[self.zero_ind],np.array([-1,0]))
        if up_loc[0]<0:
            self.up_loc = []
        else:
            self.up_loc = up_loc

    def get_child_states(self):
```

```python
        state_tile = self.state.reshape((3,3))
        future_states = []

        if len(self.left_loc) != 0:
            x_n, y_n = self.left_loc.ravel()
            left_swap = np.copy(state_tile)
            left_swap[x_n, y_n], left_swap[x_n, y_n+1] = left_swap[x_n, y_n+1], left_swap[x_n, y_n]
            future_states.append(left_swap.flatten())

        if len(self.up_loc) != 0:
            x_n, y_n = self.up_loc.ravel()
            up_swap = np.copy(state_tile)
            up_swap[x_n, y_n], up_swap[x_n+1, y_n] = up_swap[x_n+1, y_n], up_swap[x_n, y_n]
            future_states.append(up_swap.flatten())

        if len(self.right_loc) != 0:
            x_n, y_n = self.right_loc.ravel()
            right_swap = np.copy(state_tile)
            right_swap[x_n, y_n], right_swap[x_n, y_n-1] = right_swap[x_n, y_n-1], right_swap[x_n, y_n]
            future_states.append(right_swap.flatten())

        if len(self.down_loc) != 0:
            x_n, y_n = self.down_loc.ravel()
            down_swap = np.copy(state_tile)
            down_swap[x_n, y_n], down_swap[x_n-1, y_n] = down_swap[x_n-1, y_n], down_swap[x_n, y_n]
            future_states.append(down_swap.flatten())

        return future_states

##############################################################################################################################

def bfs(initial_state):

    progress_counter = 0
    root_node = node(initial_state, None, progress_counter)               #Root node creation
    queue = [root_node]                                 #Create queue to store nodes to be visited

    while len(queue)>0:

        curr_parent_node = queue.pop(0)
        next_states = curr_parent_node.get_child_states()

        for i in next_states:

            #Check if future state in visited node
            check = visit_checker(i)
            if check:
                continue

            progress_counter += 1
            curr_child_node = node(i, curr_parent_node, progress_counter)
            queue.append(curr_child_node)

            reached_goal = np.array_equal(i, goal_state)

            if reached_goal:
                print('\nFound!\n')
                back_track = []
                node_id_tracker = []
                while not curr_child_node.parent_node==None:
                    back_track.append(curr_child_node.state)
                    node_id_tracker.append(curr_child_node.node_id)
                    curr_child_node = curr_child_node.parent_node
                queue = []
                return back_track, node_id_tracker


##############################################################################################################################

def main():

    #Test Case 1

    print('Test Case I:')

    sample_state = np.array([1, 6, 7, 2, 0, 5, 4, 3, 8])

    print('\nInitial state:')
    print_matrix(sample_state)

    back_track_state, node_id_tracker = bfs(sample_state)

    with open('test_case_1_info.txt', 'w') as f1:
        f1.write(str(1))
        f1.write('\t\t')
        f1.write(np.array2string(sample_state))
        f1.write('\n')
        for i in range(len(back_track_state)-1,-1,-1):
```

```python
            print('\nStep',len(back_track_state)-i,'\n')
            print_matrix(back_track_state[i])

            f1.write(str(node_id_tracker[i]))
            f1.write('\t\t')
            f1.write(np.array2string(back_track_state[i]))
            f1.write('\n')


    #Test case 2
    sample_state = np.array([4, 5, 8, 2, 1, 5, 3, 6, 0])

    print('\n\nTest Case II:')

    print('\nInitial state:')
    print_matrix(sample_state)

    back_track_state = bfs(sample_state)

    with open('test_case_2_info.txt', 'w') as f2:
        f2.write(str(1))
        f2.write('\t\t')
        f2.write(np.array2string(sample_state))
        f2.write('\n')
        for i in range(len(back_track_state)-1,-1,-1):

            print('\nStep',len(back_track_state)-i,'\n')
            print_matrix(back_track_state[i])

            f2.write(str(node_id_tracker[i]))
            f2.write('\t\t')
            f2.write(np.array2string(back_track_state[i]))
            f2.write('\n')

########################################################################################################################################

if __name__=='__main__':
    main()
```