1

利用如下 Python 程序求解：

```python
import heapq


def heuristic(state, target):
    return sum(
        1 for i, tile in enumerate(state)
        if tile != 0 and tile != target[i]
    )


def get_neighbors(state):
    neighbors = []
    side = 3
    zero_index = state.index(0)
    row, col = divmod(zero_index, side)
    moves = []
    if row > 0:
        moves.append(-side)  # up
    if row < side - 1:
        moves.append(side)  # down
    if col > 0:
        moves.append(-1)  # left
    if col < side - 1:
        moves.append(1)  # right
    for move in moves:
        new_index = zero_index + move
        new_state = list(state)
        new_state[zero_index], new_state[new_index] = (
            new_state[new_index],
            new_state[zero_index]
        )
        neighbors.append(tuple(new_state))
    return neighbors


def a_star(start, target):
    open_set = []
    heapq.heappush(open_set, (heuristic(start, target), 0, start))
    came_from = {}
    cost_so_far = {start: 0}

    while open_set:
        f, g, current = heapq.heappop(open_set)

        # When expanding a state, show its neighbors
        for neighbor in get_neighbors(current):
            new_cost = g + 1
            if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
                cost_so_far[neighbor] = new_cost
                h = heuristic(neighbor, target)
                priority = new_cost + h
                came_from[neighbor] = current
                heapq.heappush(open_set, (priority, new_cost, neighbor))
                # Output each step
                print(f"{current} -> {neighbor}: cost {priority}")
                if neighbor == target:
                    # if found during neighbor generation, finish search
                    path = [neighbor]
                    while path[-1] in came_from:
                        path.append(came_from[path[-1]])
                    path.reverse()
                    return path
    return None


if __name__ == "__main__":
    start_state = (2, 8, 3, 1, 6, 4, 7, 0, 5)
    target_state = (1, 2, 3, 8, 0, 4, 7, 6, 5)
    solution_path = a_star(start_state, target_state)
    if solution_path:
        print("\nSolution path:")
        for s in solution_path:
            print(s)
    else:
        print("No solution found.")
```

得到如下输出：

```
(2, 8, 3, 1, 6, 4, 7, 0, 5) -> (2, 8, 3, 1, 0, 4, 7, 6, 5): cost 4
(2, 8, 3, 1, 6, 4, 7, 0, 5) -> (2, 8, 3, 1, 6, 4, 0, 7, 5): cost 6
(2, 8, 3, 1, 6, 4, 7, 0, 5) -> (2, 8, 3, 1, 6, 4, 7, 5, 0): cost 6
(2, 8, 3, 1, 0, 4, 7, 6, 5) -> (2, 0, 3, 1, 8, 4, 7, 6, 5): cost 5
(2, 8, 3, 1, 0, 4, 7, 6, 5) -> (2, 8, 3, 0, 1, 4, 7, 6, 5): cost 5
(2, 8, 3, 1, 0, 4, 7, 6, 5) -> (2, 8, 3, 1, 4, 0, 7, 6, 5): cost 6
(2, 0, 3, 1, 8, 4, 7, 6, 5) -> (0, 2, 3, 1, 8, 4, 7, 6, 5): cost 5
(2, 0, 3, 1, 8, 4, 7, 6, 5) -> (2, 3, 0, 1, 8, 4, 7, 6, 5): cost 7
(2, 8, 3, 0, 1, 4, 7, 6, 5) -> (0, 8, 3, 2, 1, 4, 7, 6, 5): cost 6
(2, 8, 3, 0, 1, 4, 7, 6, 5) -> (2, 8, 3, 7, 1, 4, 0, 6, 5): cost 7
(0, 2, 3, 1, 8, 4, 7, 6, 5) -> (1, 2, 3, 0, 8, 4, 7, 6, 5): cost 5
(1, 2, 3, 0, 8, 4, 7, 6, 5) -> (1, 2, 3, 7, 8, 4, 0, 6, 5): cost 7
(1, 2, 3, 0, 8, 4, 7, 6, 5) -> (1, 2, 3, 8, 0, 4, 7, 6, 5): cost 5

Solution path:
(2, 8, 3, 1, 6, 4, 7, 0, 5)
(2, 8, 3, 1, 0, 4, 7, 6, 5)
(2, 0, 3, 1, 8, 4, 7, 6, 5)
(0, 2, 3, 1, 8, 4, 7, 6, 5)
(1, 2, 3, 0, 8, 4, 7, 6, 5)
(1, 2, 3, 8, 0, 4, 7, 6, 5)
```

根据输出结果容易画出搜索树，故此处略。

2

同上，只需修改启发式函数和始末状态：

```
import heapq


def heuristic(state, target):
    n = int(len(state) ** 0.5)
        target_positions = {tile: divmod(i, n) for i, tile in enumerate(target)}
        return sum(
            abs(r - target_positions[tile][0]) + abs(c - target_positions[tile]
[1])
            for i, tile in enumerate(state)
            if tile != 0
            for r, c in [divmod(i, n)]
        )

# ...

if __name__ == "__main__":
    start_state = (1, 2, 3, 0, 8, 4, 7, 6, 5)
    target_state = (2, 8, 3, 1, 6, 4, 7, 0, 5)

    # ...
```

输出结果为：

```
(1, 2, 3, 0, 8, 4, 7, 6, 5) -> (0, 2, 3, 1, 8, 4, 7, 6, 5): cost 4
(1, 2, 3, 0, 8, 4, 7, 6, 5) -> (1, 2, 3, 7, 8, 4, 0, 6, 5): cost 6
(1, 2, 3, 0, 8, 4, 7, 6, 5) -> (1, 2, 3, 8, 0, 4, 7, 6, 5): cost 6
(0, 2, 3, 1, 8, 4, 7, 6, 5) -> (2, 0, 3, 1, 8, 4, 7, 6, 5): cost 4
(2, 0, 3, 1, 8, 4, 7, 6, 5) -> (2, 8, 3, 1, 0, 4, 7, 6, 5): cost 4
(2, 0, 3, 1, 8, 4, 7, 6, 5) -> (2, 3, 0, 1, 8, 4, 7, 6, 5): cost 6
(2, 8, 3, 1, 0, 4, 7, 6, 5) -> (2, 8, 3, 1, 6, 4, 7, 0, 5): cost 4

Solution path:
(1, 2, 3, 0, 8, 4, 7, 6, 5)
(0, 2, 3, 1, 8, 4, 7, 6, 5)
(2, 0, 3, 1, 8, 4, 7, 6, 5)
(2, 8, 3, 1, 0, 4, 7, 6, 5)
(2, 8, 3, 1, 6, 4, 7, 0, 5)
```

3

模拟带 $\alpha-\beta$ 剪枝的 `min-max` 搜索算法得根节点值为 4，需要剪枝的节点为（路径表示法，左路优先，从零开始）:

```
001
002
01
100
102
110
200
201
211
220
221
```

共 11 个节点。