DFS Branch and Bound (8 Puzzle Problem)

This document contains the implementation of the 8 Puzzle problem using the Depth First Branch and Bound (DFS B&B;) algorithm in Python.

```
import copy

GOAL_STATE = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

MOVES = [(-1, 0), (1, 0), (0, -1), (0, 1)]


def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            tile = state[i][j]
            if tile != 0:
                goal_x, goal_y = divmod(tile - 1, 3)
                distance += abs(i - goal_x) + abs(j - goal_y)
    return distance


def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
    return -1, -1


def successors(state):
    blank_i, blank_j = find_blank(state)
    for di, dj in MOVES:
        ni, nj = blank_i + di, blank_j + dj
        if 0 <= ni < 3 and 0 <= nj < 3:
            new_state = copy.deepcopy(state)
            new_state[blank_i][blank_j], new_state[ni][nj] = new_state[ni][nj], new_state[blank_i][blank
            yield new_state


def is_goal(state):
    return state == GOAL_STATE


def state_to_tuple(state):
    return tuple(tuple(row) for row in state)


def is_solvable(state):
    flat = [tile for row in state for tile in row if tile != 0]
    inversions = 0
    for i in range(len(flat)):
        for j in range(i + 1, len(flat)):
            if flat[i] > flat[j]:
                inversions += 1
                print(flat[i],flat[j])
    print(inversions)
    return inversions % 2 == 0


def dfbnb(start_state):
    if not is_solvable(start_state):
        return None
```

```
    bound = float('inf')
    best_path = None

    stack = []
    visited = dict()

    stack.append((start_state, [], 0))

    while stack:
        state, path, cost = stack.pop()
        state_t = state_to_tuple(state)

        heuristic = manhattan_distance(state)
        total_cost = cost + heuristic

        if total_cost >= bound:
            continue

        if (state_t in visited) and (visited[state_t] <= cost):
            continue
        visited[state_t] = cost

        if is_goal(state):
            bound = cost
            print(bound)
            best_path = path + [state]
            continue

        for succ in successors(state):
            stack.append((succ, path + [state], cost + 1))

    return best_path


def print_solution(solution):
    if not solution:
        print("No solution found or puzzle is unsolvable.")
        return
    for idx, state in enumerate(solution):
        print(f"Step {idx}:")
        for row in state:
            print(row)
        print("---")
    print(f"Total moves: {len(solution) - 1}")


if __name__ == "__main__":
    start = [
        [7, 2, 3],
        [5, 0, 6],
        [4, 1, 8]
    ]

    solution = dfbnb(start)
    print_solution(solution)
```