Lab-3: 8 puzzle problems using DFS and Manhattan distance

Observation book:

classmate
Date 08/10/2024
Page ____

8- Puzzle :

~~DFS: Manhattan Distance:~~ ~~DFS~~: Manhattan Distance:

~~Algorithm :~~

~~* Initialize the goal array in a function.~~
~~- This function returns True if current state matches the goal. Else false.~~

~~* Define 4 functions: Move left, move right, move up, move down.~~

~~* Identify the position of the blank tile.~~
~~- If corner, make a move for the neighbouring 2 tiles.~~
~~- If center, make a move for the neighbouring 4 Tiles.~~
~~- Else, make a move for the neighbouring 3 Tiles.~~

~~* For every move, check the Manhattan distance. If it is~~

Algorithm :

* Represent the puzzle state as a list where 0 represents the empty space.
* Define a function to check if the current state matches the goal state.
* Define a function to calculate the Manhattan Distance for a given state.
* Define a function to return a list of new

states and their updated positions upon making a move.

* Use a stack to store the states after each move.
* Compare the top element of this stack with the ~~final~~ goal state.
  - If it is a match, then pop the states one by one and add it to the path array.
    - Reverse the path array to print it.
  - If not a match, make the next move and add the new state onto the stack.

* Maintain uniqueness of the states to avoid an infinite loop.

Check with DFS

Proof

<u>Code</u>:

```
import copy

directions = [(-1,0), (1,0), (0,-1), (0,1)]
goal = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 0]]

def Manhattan(state):
    d = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                gx, gy = divmod(state[i][j]-1, 3)
                d += abs(i-gx) + abs(j-gy)
    return d
```

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

def goal(state):
    return state == goal

def print_board(state):
    for row in state:
        print(row)
    print("\n")

def dfs(state, depth, moves):
    bx, by = find_blank(state)
    if goal(state):
        return True, state, moves
    if depth == 0:
        return False, None, moves
    possible_moves = []
    for dx, dy in directions:
        nx, ny = bx + dx, by + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = copy.deepcopy(state)
            new_state[bx][by], new_state[nx][ny] =
                new_state[nx][ny], new_state[bx][by]
            md = Manhattan(new_state)
            possible_moves.append((md, new_state))
    possible_moves.sort(key = lambda x: x[0])
    for _, next_state in possible_moves:
        moves.append(next_state)
        print("Move made: ")
```

```
        print_board(next_state)
        found, result, moves = dfs(next_state, depth-1,
                                   moves)
        if found:
            return True, result, moves
        moves.pop()
    return False, None, moves

def solve(initial, depth = 30):
    moves = [initial]
    print("Initial State:")
    print_board(initial)
    found, final, moves = dfs(initial, depth, moves)
    if found:
        print("Solution Found!")
        print("Final state:")
        print_board(final)
    else:
        print("No solution found within the depth limit")

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

solve(initial)
```

Output:

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

Date
Page

Move made:
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

Move made:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Solution found!
Final state:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

19/11

Output:

```
Enter row 1: 1 0 3
Enter row 2: 4 2 6
Enter row 3: 7 5 8
Solution found:
1 0 3
4 2 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Nikhilesh C 1BM22CS181
```