Lab-3 8-puzzle problem using DFS and MD

→ 1. ~~Using DFS~~ Initial Setup

We assume initial array and final array (both 3x3) are defined with random integer from 0-8 placed (0 = empty space)

→ Algorithm using DFS:

```
# def dfs (start, end):
        Stack = []
        Stack.append (hash(start))
        while Stack[-1] != end:
            valid-moves = getMoves(stack[-1])
Stack = [] for move in valid-moves:
def dfs (start, end, zeroPos):
        Stack = []; visited = []
        Stack.append (start)
        while Stack[-1] != end:
            valid-moves = getMoves(stack[-1])
            for move in valid-moves:
                # applyMove
                new-board = board.copy()
                swap (new board[zeroPos],
                      new-board[zeroPos + move])
                if new-board not in visited:
                    stack.append(new-board)
                    visited.append(new-board)
                else:
                    while stack.pop() != new-board:
                        continue
                continue
        return stack[-1];
```

Utility functions:

- def getMoves(board, zeroPos):
    valid_moves = [(0,1), (1,0), (0,-1), (-1,0)]
    for move in valid_moves:
        if 0 <= zeroPos[0] + move[0] <= 3
        and 0 <= zeroPos[1] + move[1] <= 3:
            continue
        else:
            valid_moves.remove(move)

→ Using Manhattan Distance:

def md(start, end, zeroPos):
    queue = []; visited = []
    queue.append(start)
    visited.append(start)
    valid_moves = getMoves(board, zeroPos)
    while queue[0] != end:
        for move in valid_moves:
            new_board = board.copy()
            swap(new_board[zeroPos], new_board[zeroPos])
            if new_board not in visited:
                queue.append(new_board)
                visited.append(new_board)
                queue.sort(key = getMD())
        queue.pop(0)

→ Utility functions:

```
- def getMD(board):
      c = 0
      for i in range(3)
          for j in range(3):
              c += (i - board[i][j]//3)
                 + (j - board[i][j]%3)
      return c
```

Eg:

Start = 2 3 4 | 5 7 1 | 8 6 0 MD = 19

Possible Moves =

① 2 3 4 | 5 7 1 | 8 0 6 MD = 19

② 2 3 4 | 5 7 0 | 8 6 1 MD = 19

Go with ①

Possible Moves:

2 3 4 | 5 7 1 | 0 8 6 MD = 17

2 3 4 | 5 0 1 | 8 7 6 MD = 17

X 2 3 4 | 5 7 1 | 8 6 0

Go with ①

Program:

```
import heapq
class Puzzle:
    def __init__(self):
        self.board = [
            [1,2,3]
            [4,0,5]
            [7,8,6] ]
        self.end = [
            [0,1,2]   [1,2,3]
            [3,4,5]   [4,5,6]
            [6,7,8]   [7,8,0] ]

    def getMoves(self, board):
        zero_pos = self.zero_index(board)
        moves = [(0,1), (1,0), (0,-1), (-1,0)]
        valid_moves = []
        for move in moves:
            if 0<= zero_pos[0] + move[0] < 3 and
                0 <= zero_pos[1] + move[1] < 3:
                valid_moves.append(move)
        return valid_moves

    def zero_index(self, board):
        for i in range(3)
            for j in range(3):
                if board[i][j] == 0:
                    return (i, j)

    def hash(self, board):
        return tuple(map(tuple, board))
```

```
def display(self, board):
    for ls in board:
        print(*ls)

def dfs(self):
    stack = []
    visited = []
    stack.append(self.board)
    visited.append(self.bhash(self.board))
    while stack:
        top = stack[-1]
        if self.bhash(top) == self.bhash(self.end):
            break
        valid_moves = self.getMoves(top)
        added = False
        for move in valid_moves:
            new_board = [row[:] for row in top]
            zeroPos = self.zero_index(new_board)
            newPos = [zeroPos[0] + move[0],
                      zeroPos[1] + move[1]]
            new_board[newPos[0]][newPos[1]],
            new_board[zeroPos[0]][zeroPos[1]], =
            new_board[zeroPos[0]][zeroPos[1]],
            new_board[newPos[0]][newPos[1]]
            if self.bhash(new_board) not in visited:
                stack.append(new_board)
                visited.append(self.bhash(new_board))
                added = True
                break
        if not added:
            stack.pop()
    while stack:
        self.display(stack.pop(0)); print("---")
```

```
c= Puzzle()
print("DFS: ")
c.dfs()
```

Output:

DFS:

```
1 2 3
4 0 5
7 8 6
-----
1 2 3
4 5 0
7 8 6
-----
1 2 3
4 5 6
7 8 0
```