LAB-3

8/10/24

* Write an algorithm and a program to solve 8-puzzle game

~~def Man~~

Step 1 :- Initialize goal state and possible moves

goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, _]]

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

Step 2 :- Function to calculate Manhattan distance

```
def manhattan_dist(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] != '_':
                goal_i, goal_j = divmod(state[i][j]-1, 3)
                distance += abs(i-goal_i)+ abs(j-goal_j)
return distance
```

(6-1, 3) $\frac{1}{3}$

(1,3)

(2, 2)

Step 3 :- Check if current state is equal to goal state

```
def completed(state):
    return goal_state == state
```

Step 4 :- Check all possible moves

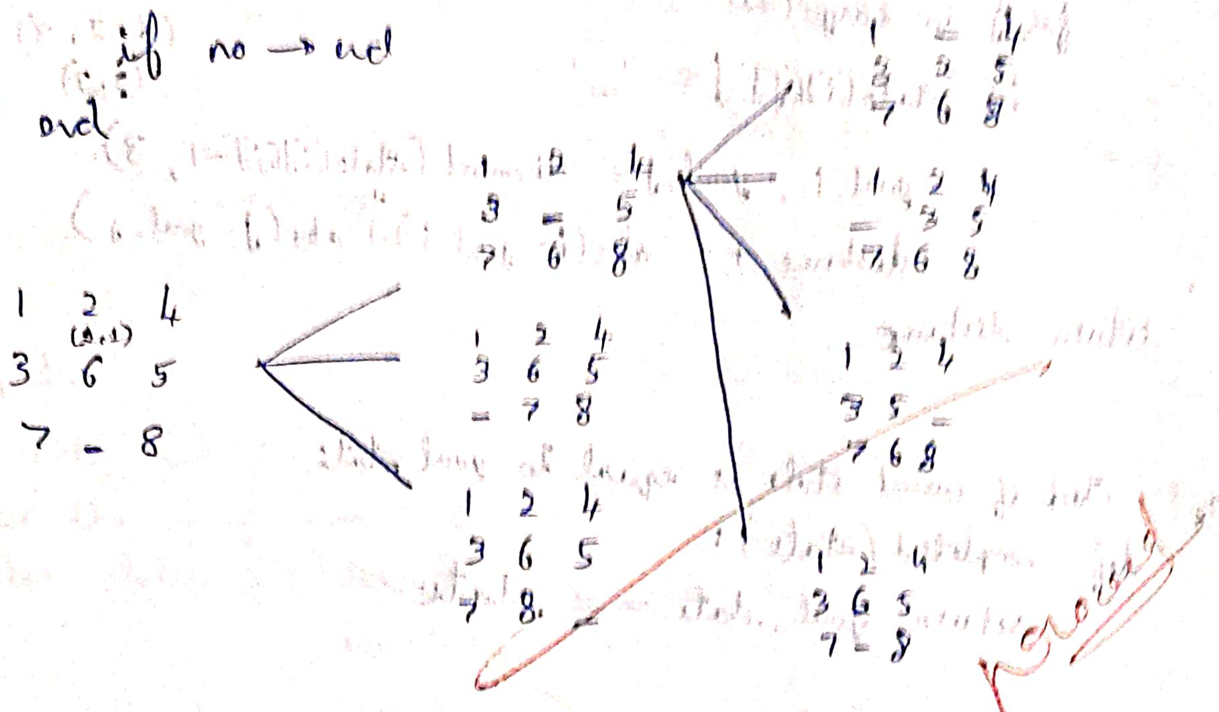
```
def neighbours (state):
    for i in range(3):
        for j in range(3):
```

if state[i][j] == '_' => check all 4 directions using the moves matrix, you can find that

Return the neighbors array.

Step 5:-

```
def dfs (state):
    queue = deque([(state, [state])])
    visited = set()
    while queue:
        current_state, path = queue.popleft()
# check if current state is goal state
# skip visited states
  if not visited, add to queue
  get the neighbours
  if no → end
end
```



1	2	3 (1,2)
4	5	6
7	8	_

Code :-

```
from collections import deque

GOAL_STATE = [[1, 2, 3], [4, 5, 6], [7, 8, '-']]
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):
            if state [i][j] != '-':
                goal_i, goal_j = divmod (state [i][j] - 1, 3)
                distance += abs(i - goal_i) + abs(j - goal_j)
    return distance

def is_goal_state (state):
    return state == GOAL_STATE

def get_neighbors (state):
    neighbors = []
    for i in range (3):
        for j in range (3):
            if state [i][j] == '-' :
                for move in MOVES:
                    new_i , new_j = i+move[0], j + move[1]
                    if 0< = new_i < 3 and 0 < = new_j < 3 :
                        new_state = [row [:] for row in state ]
                        new_state [i][j], new_state [new_i][new_j] =
                                new_state [new_i] [new_j], new_state[i][j]
                        neighbours. append (new_state)
    return neighbors

def dfs (state):
    queue = deque ([(state, [state])])
    visited = set ()
```

```python
while queue:
    current_state, path = queue.popleft()
    if is_goal_state(current_state):
        return path
    if tuple(map(tuple, current_state)) in visited:
        continue
    visited.add(tuple(map(tuple, current_state)))
    for neighbor in get_neighbors(current_state):
        queue.append((neighbor, path + [neighbor]))

return None

initial_state = [[4, 1, 3], [7, 2, 6], [5, 8, '-']]
path = dfs(initial_state)
if path:
    print("Solution found:")
    for state in path:
        for row in state:
            print(row)
        print()
else:
    print("No solution")
```

<u>output</u>:

$$\begin{bmatrix} 4 & 1 & 3 \\ 7 & 2 & 6 \\ 5 & 8 & - \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 1 & 3 \\ 7 & 2 & 6 \\ 5 & - & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 1 & 3 \\ 7 & 2 & 6 \\ - & 5 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 1 \\ - & 2 \\ 7 & 5 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & - & 8 \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & - & 6 \\ 7 & 5 & 8 \end{bmatrix} \leftarrow \begin{bmatrix} 1 & - & 3 \\ 4 & 2 & 6 \\ 7 & 5 & 8 \end{bmatrix} \leftarrow \begin{bmatrix} - & 1 & 3 \\ 4 & 2 & 6 \\ 7 & 5 & 8 \end{bmatrix}$$

$$\hookrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & - \end{bmatrix}$$

8/10/24