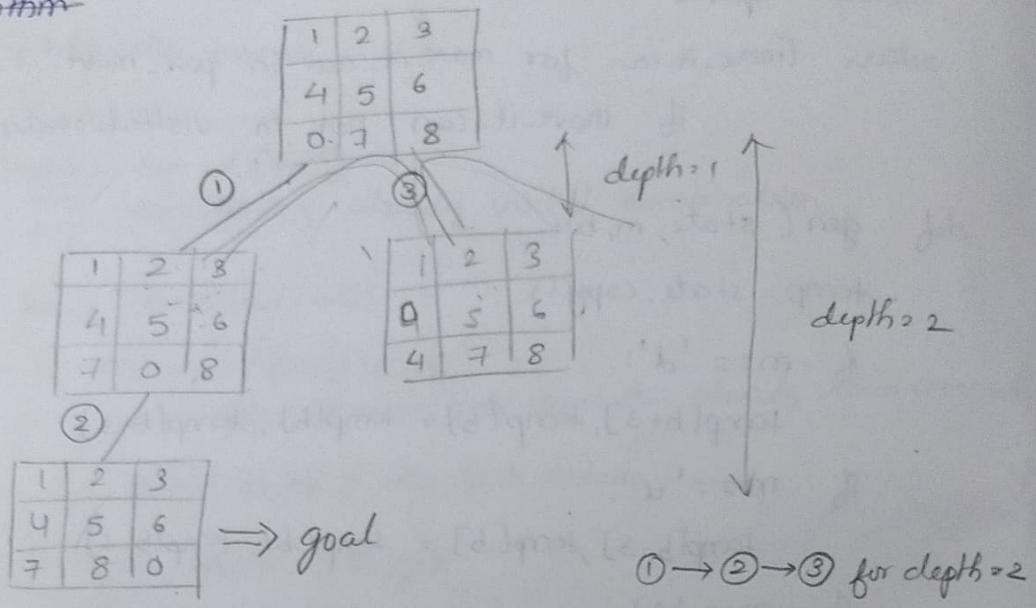
8/12/2021 8/12/23

8-puzzle iterative deepening search algorithm

~~Algorithm~~

1)



Algorithm:

1) Initialize the initial state = [] and goal state for the 8puzzle

goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0] # 0 is the blank space

2) Set the depth = 1 and expand the initial state

The depth_limited_search (depth) is performed

```
if node.State = goal
    return node
else if depth=0:
    return None
else
    for neighbour in get_neighbours(state):
        child = puzzlenode (neighbour, node)
        result = depth_limited_search (depth-1)
        if result == True:
            return result
```

3) After one iteration where depth=1, increment the depth by 1 and perform depth_limited_search again

4) Here get_neighbours will generate the possible moves ~~in the same level~~ by swapping the '0' tile

5) The path traversed is printed to reach the goal state

Code:

```
def id_dfs(puzzle, goal, get_moves):
    import itertools

    def dfs(route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in get_moves(route[-1]):
            if move not in route:
                next_route = dfs(route + [move], depth - 1)
                if next_route:
                    return next_route

    for depth in itertools.count():
        route = dfs([puzzle], depth)
        if route:
            return route

def possible_moves(state):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
        d.append('u')
    if b not in [6,7,8]:
        d.append('d')
    if b not in [0,3,6]:
        d.append('l')
    if b not in [2,5,8]:
        d.append('r')
    pos_moves = []
    for i in d:
        pos_moves.append(generate(state,i,b))
    return pos_moves
```

```
def generate(state, m, b):
    temp = state.copy()
    if m=='d':
        temp[b+3], temp[b] = temp[b], temp[b+3]
    if m=='u':
        temp[b-3], temp[b] = temp[b], temp[b-3]
    if m=='l':
        temp[b-1], temp[b] = temp[b], temp[b-1]
    if m=='r':
        temp[b+1], temp[b] = temp[b], temp[b+1]
    return temp

initial = [1, 2, 3, 0, 4, 6, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
route = id_dfs(initial, goal, possible_moves)
if route:
    print("Success!! It is possible to solve 8 puzzle problem")
    print("Path:", route)
else:
    print("Failed to find a solution")
```

Output:

Success!! It is possible to solve 8 puzzle problem

Path: [[1, 2, 3, 0, 4, 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]]

```
Success!! It is possible to solve 8 Puzzle problem
Path: [1, 2, 3, 0, 4, 6, 7, 5, 8] [1, 2, 3, 4, 0, 6, 7, 5, 8]
Path: [1, 2, 3, 4, 5, 6, 7, 0, 8] [1, 2, 3, 4, 5, 6, 7, 8, 0]
```