24/11/23

LAB - 2

8 Puzzle BFS Algorithm

Initial State:

1	2	3
	4	6
7	5	8

Goal State

1	2	3
4	5	6
7	8	

Rules:

- The empty tile can move in 4 directions (i) up (ii) down (iii) right (iv) left.
- Cannot move diagonally.
- Can take one step at a time.

0	x	0
x	#	x
0	x	0

0 - 2 possible moves
x - 3 possible moves
- 4 possible moves.

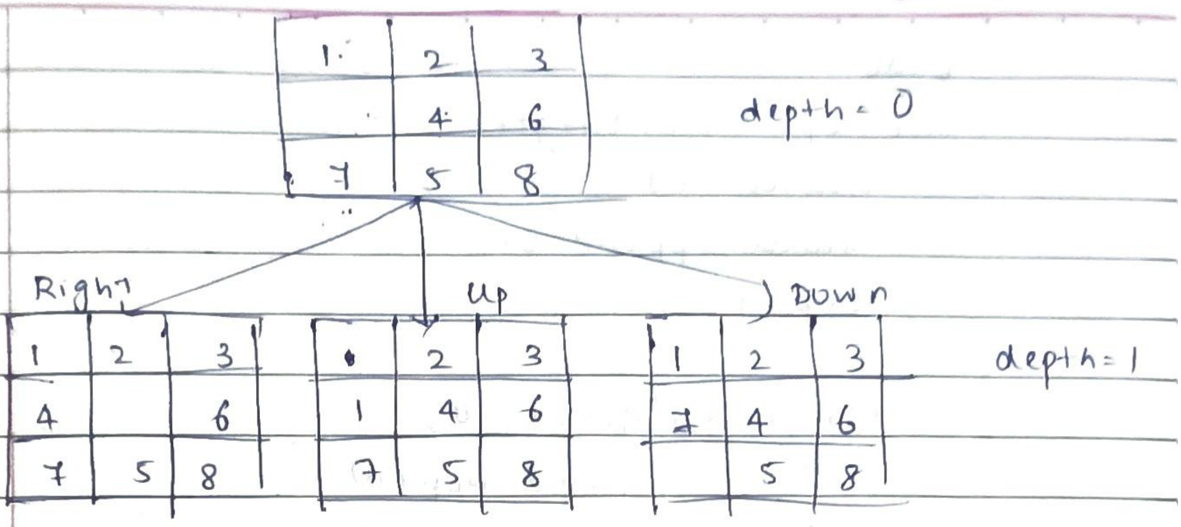
Breadth first Algorithm: (Uninformed / Non-heuristic approach).

→ Complexity O(b^d) where
b - branching factor.
d - depth factor.

For 8 puzzle problem:

$$\text{Branching factor } b = \frac{\text{all possible moves of empty tile at each position}}{\text{No of tiles}}$$

$$= \frac{24}{9} \approx 3$$



ALGORITHM:

1. Initialize Queue and Explored Set:
2. BFS loop:
   → Dequeue a state from the front of the queue.
   → Add the dequeued state to the visited set.
   → Check if the dequeued state is target.
   If yes, print "success" and return.
   → Generate possible moves from the current state which is not visited.
   → Enqueue these state to queue.

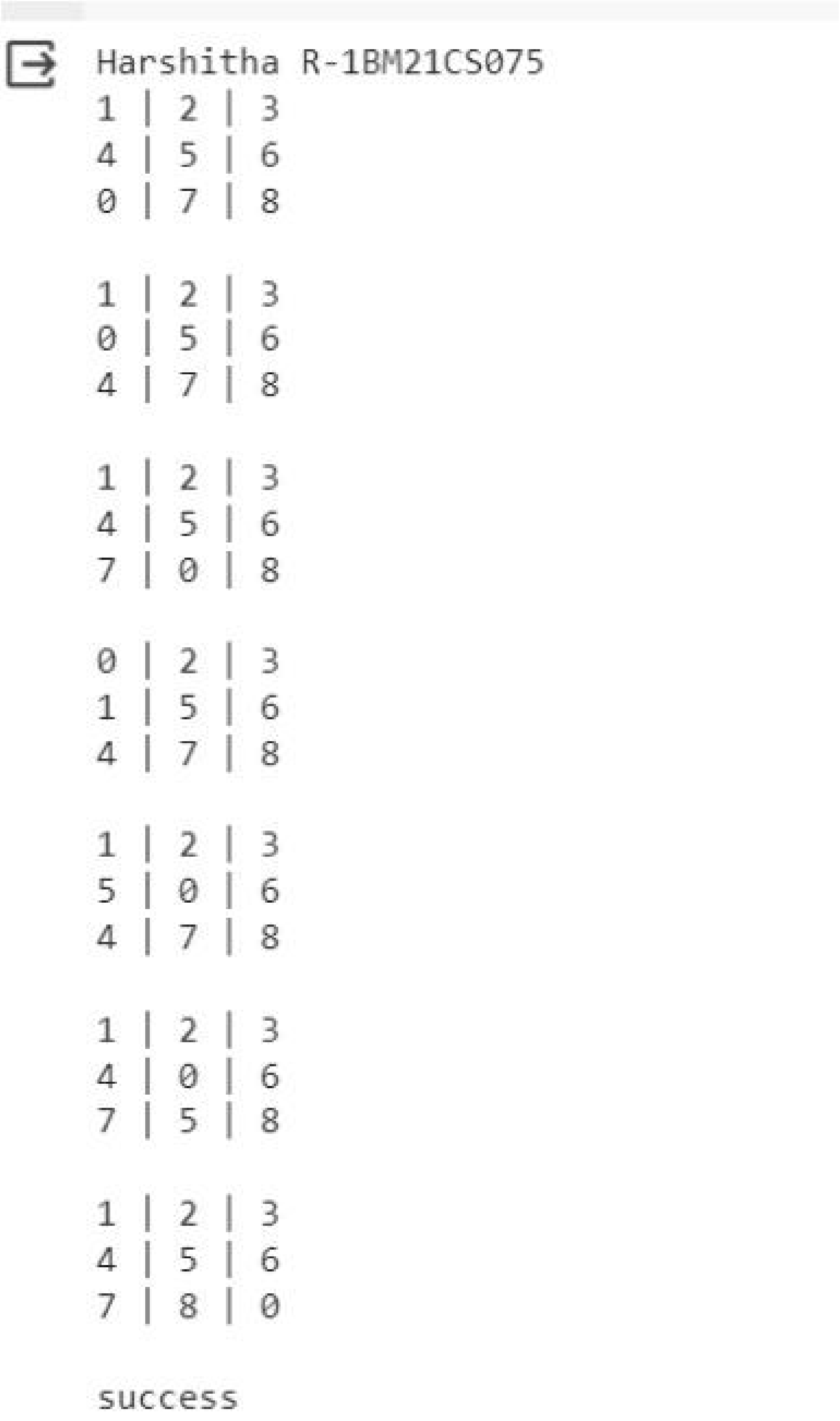
3. Possible Moves Generation:
   → Find the index of the empty spot.
   → Initialize an empty array to store possible directions.
   → Check for possible moves → up, down, left, right.
   → Generate states based on the possible moves on the current state.

4. Move Generation.
   → Create a copy of the current state.
   → Depending on the specified move, swap the empty spot with the adjacent tile.
   → Return the new state after the move.

Code:

```
def bfs (src, target):
    queue = []
    queue.append(src)
    exp = []

    while len(queue)>0;
        source = queue.pop(0)
        exp.append (source)
        print (source)

        if source == target:
            print ("success")
            return

        poss_move_to_do = []
        poss_moves_to_do = possible_moves (source,exp

        for move in poss_moves_to_do:
            if move not in exp and move not in queu
                queue.append (move)

def possible_moves( state, visited_states):
    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_it_can = []
    for i in d:
        pos_moves_it_can.append(gen(state, i, b))
    return [move_it_can for move_it_can in
              pos_moves_it_can if move_it_can
              not in visited_states]

def gen(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.

src = [1, 2, 3, 4, 5, 6, 0, 7, 8]
target = [1, 2, 3, 4, 5, 6, 7, 8, 0]
bfs (src, target)
```

24/1



```
Harshitha R-1BM21CS075
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8

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

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

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

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

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

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

success
```