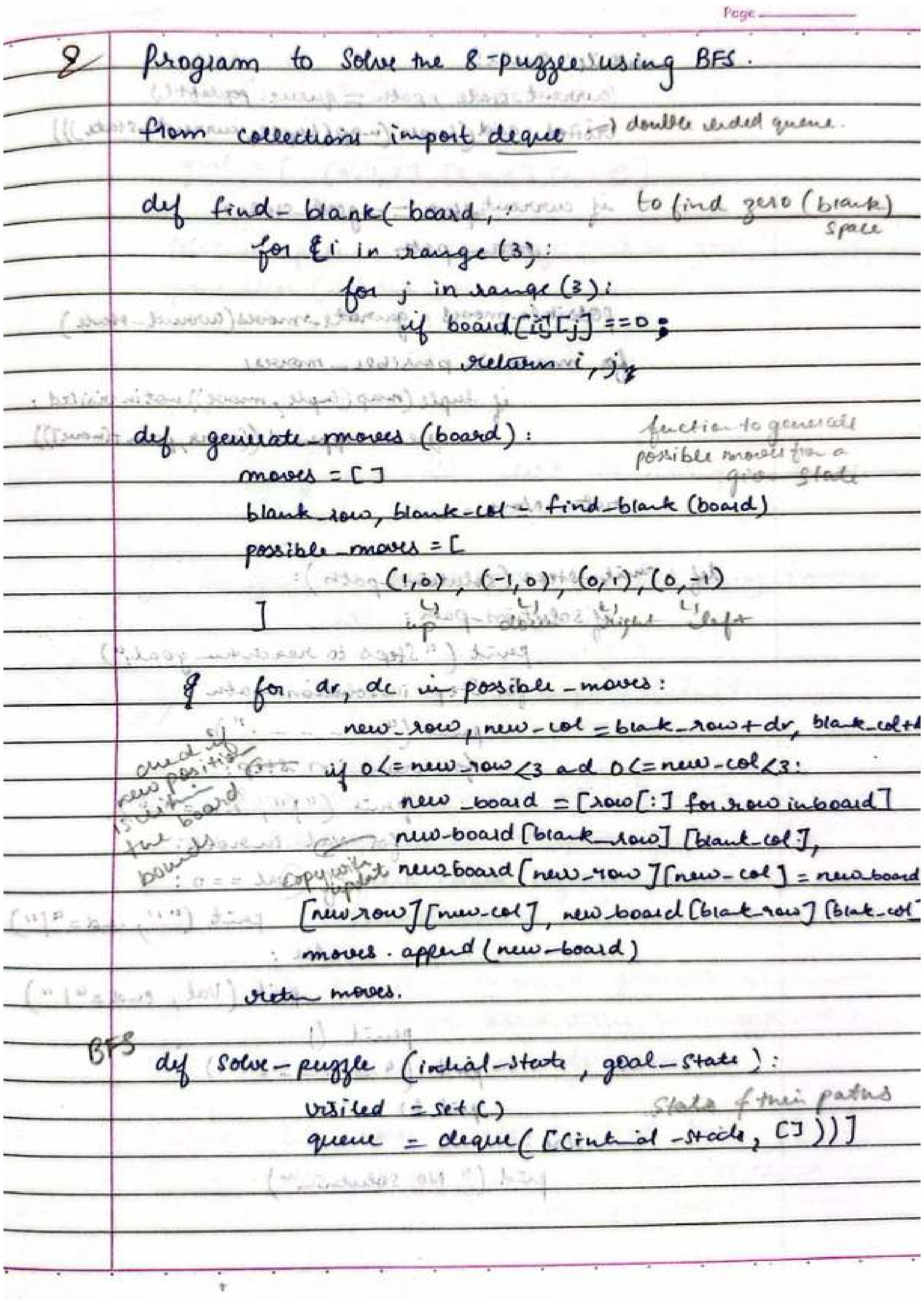
8 puzzle bfs
Code:

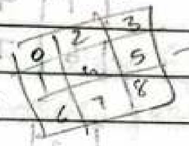


```
8  Program to Solve the 8-puzzle using BFS.

from collections import deque  → double ended queue.

def find-blank(board,                      to find zero (blank)
                                                        space
    for i in range(3):
        for j in range(3):
            if board[i][j]==0:
                return i, j

def generate-moves (board):                function to generate
                                           possible moves for a
                                                 given state
    moves = []
    blank_row, blank-col = find-blank (board)
    possible_moves = [
            (1,0), (-1,0), (0,1), (0,-1)
                 up     down   right   left
    ]

    for dr, dc in possible_moves:
        new_row, new-col = blank_row+dr, blank_col+d
check if
new position    if 0<=new_row<3 and 0<=new-col<3:
is within the        new_board = [row[:] for row in board]
board                new-board [blank_row] [blank_col],
bounds     copy with new board [new_row][new-col] = new board
           update    [new_row][new-col], new_board[blank_row][blank_col]
        moves.append(new-board)
    return moves.

BFS
def solve-puzzle (initial-state, goal-state):
        visited = set()                    state of their paths
        queue = deque([[initial-state, []])]
```

5

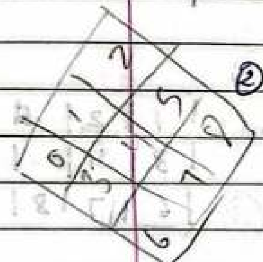
```
    while queue:
        current_state, path = queue.popleft()
        visited.add(tuple(map(tuple, current_state)))

        if current_state == goal_state:
            return path

        possible_moves = generate_moves(current_state)
        for move in possible_moves:
                if tuple(map(tuple, move)) not in visited:
                    queue.append((move, path+[move]))

    return None

def print_steps(solution_path):
    if solution_path:
            print("Steps to reach the goal:")
            for step in solution_path:
                print("_ _ _ _ .")
                for row in step:
                    print("|", end=" ")
                    for val in row:
                        if val == 0:
                            print(" ", end="|")
                        else:
                            print(val, end="|")
                    print()
                print("_ _ _ _ _")
                print()
    else:
        print("No solution")
```

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

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

solution_path = solve_puzzle(initial, goal)
print_steps (solution_path)
```

 → [0, 1, 2, 3, 4, 5, 6, 7, 8]

1	2	3
0	4	5
6	7	8

① we'll be using a doubleend queue



② find the blankspace here (0) using function

```
for i in range 3:
    for j in range3
        if board[i][j]==0 :
            ret i, j
```

③ Now generate moves using function ie move the blankspace in all possible ways ↑, ↓, ←, →

ⓘ first check if new position is in board bound (3×3)

ⓘⓘ if it is in bound generate the move ie move blank space with adjacent tiles and board for path.

Use bfs now use visited list every time generate a copy and check with goalstate.

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

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

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

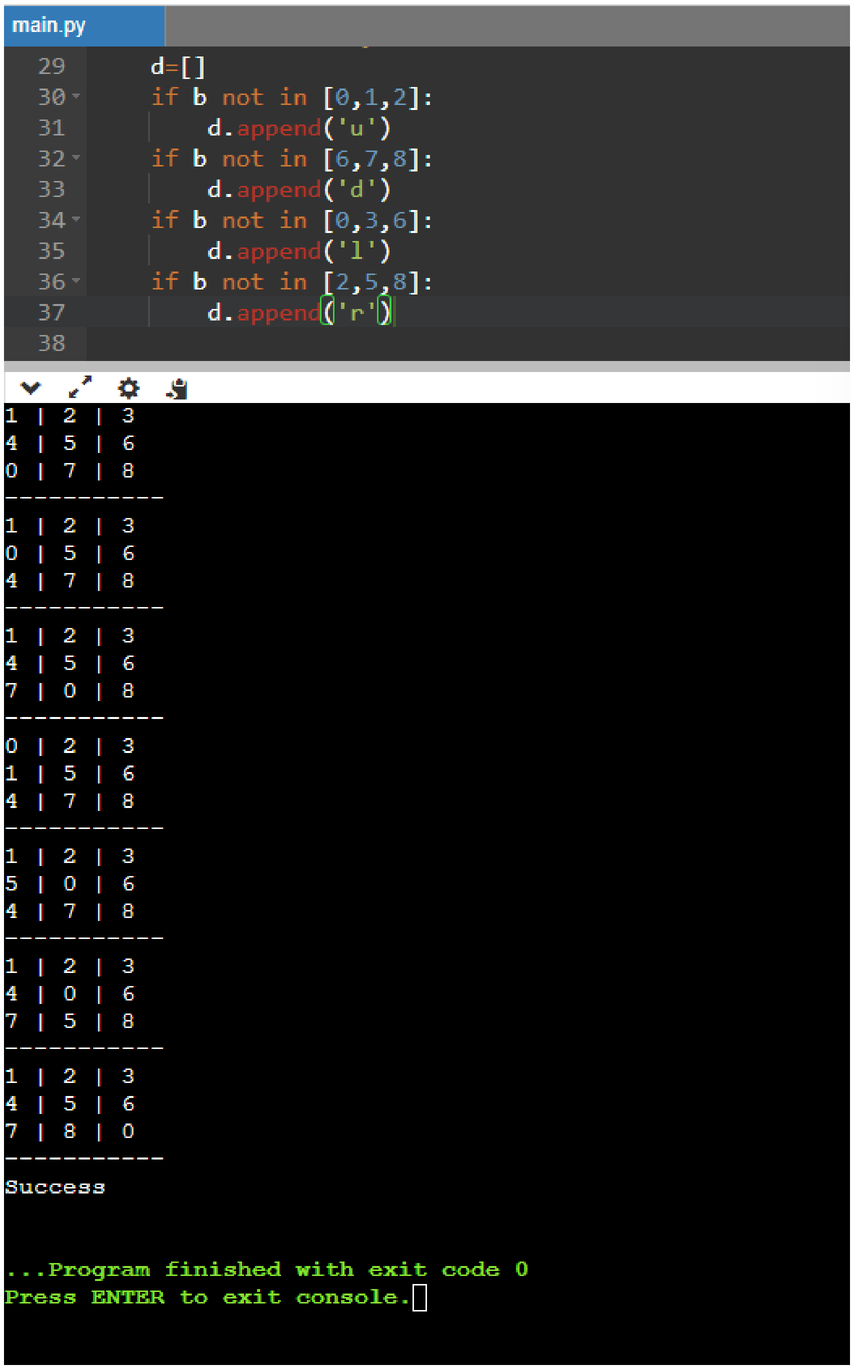
1	2	5	1	2	5	1	2	5
	3	4	3		4	3	4	
6	7	8	6	7	8	6	7	8

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

Write this Page

[illegible] 24/11

output:-



```
    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')

```

```
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


...Program finished with exit code 0
Press ENTER to exit console.
```