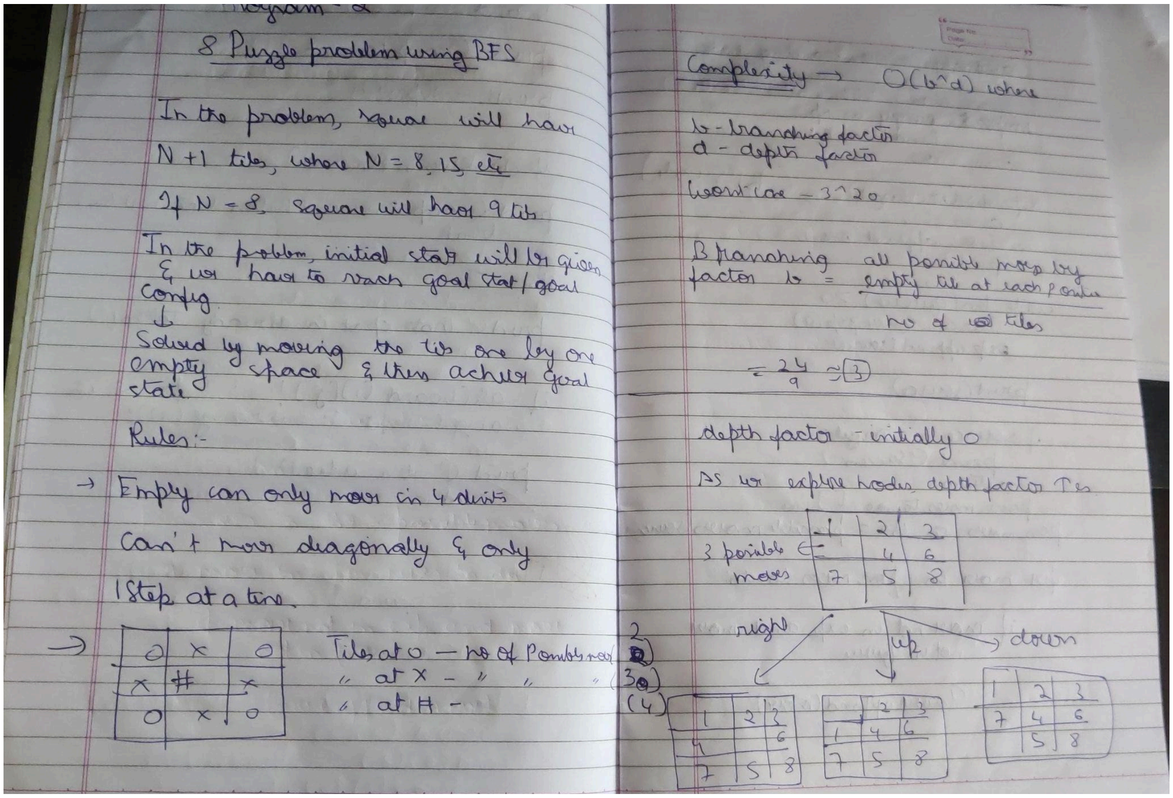


Program - 2

8 Puzzle problem using BFS

In the problem, square will have N+1 tiles, where N = 8, 15, etc

If N = 8, square will have 9 tiles

In the problem, initial state will be given & we have to reach goal state / goal config
↓
Solved by moving the tiles one by one empty space & thus achieve goal state

Rules:-

→ Empty can only move in 4 dirns

Can't move diagonally & only 1 step at a time.

→

o	x	o
x	#	x
o	x	o

Tiles at o — no of Possible moves (2)
" at x — " " " (3)
" at # — (4)

Complexity → O(b^d) where

b - branching factor
d - depth factor

Worst case - 3^20

Branching factor b = all possible moves by empty tile at each position / no of tiles

= 24/9 ≃ $\boxed{3}$

depth factor - initially 0

As we explore nodes depth factor ↑es

3 possible moves ←

1	2	3
	4	6
7	5	8

right

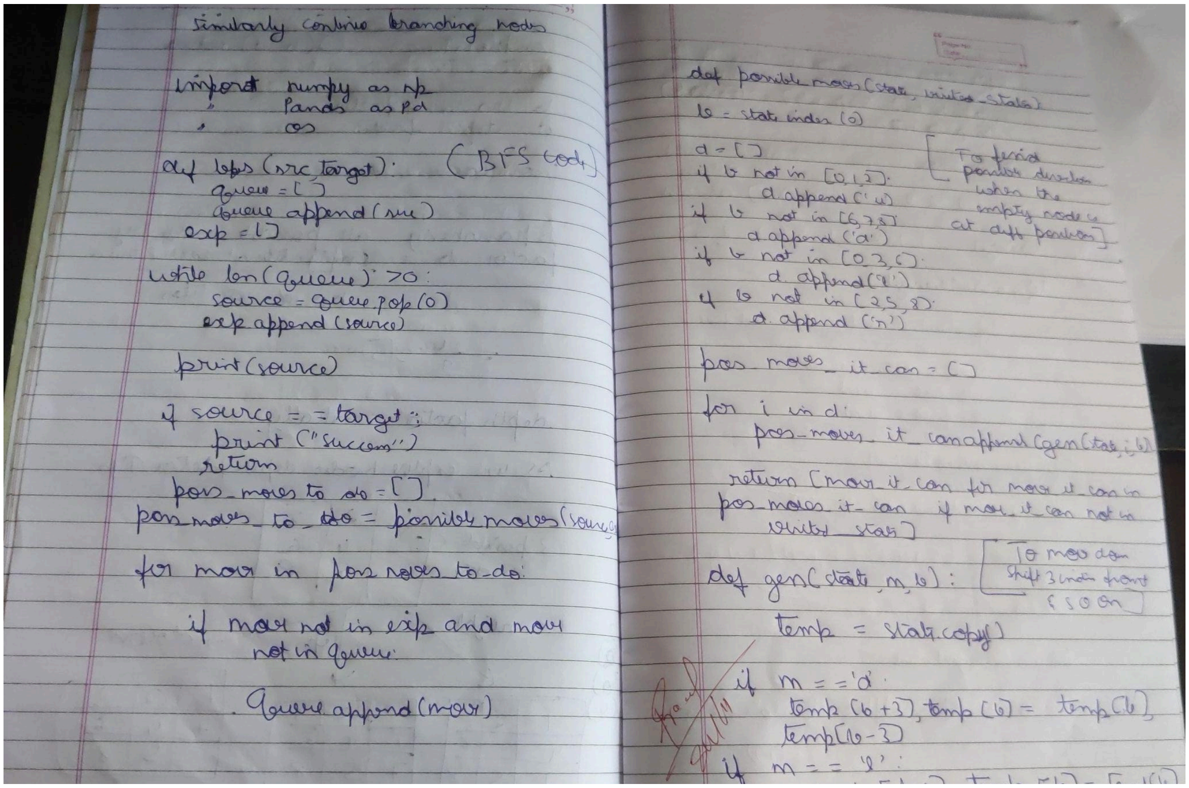
1	2	3
4		6
7	5	8

up

	2	3
1	4	6
7	5	8

down

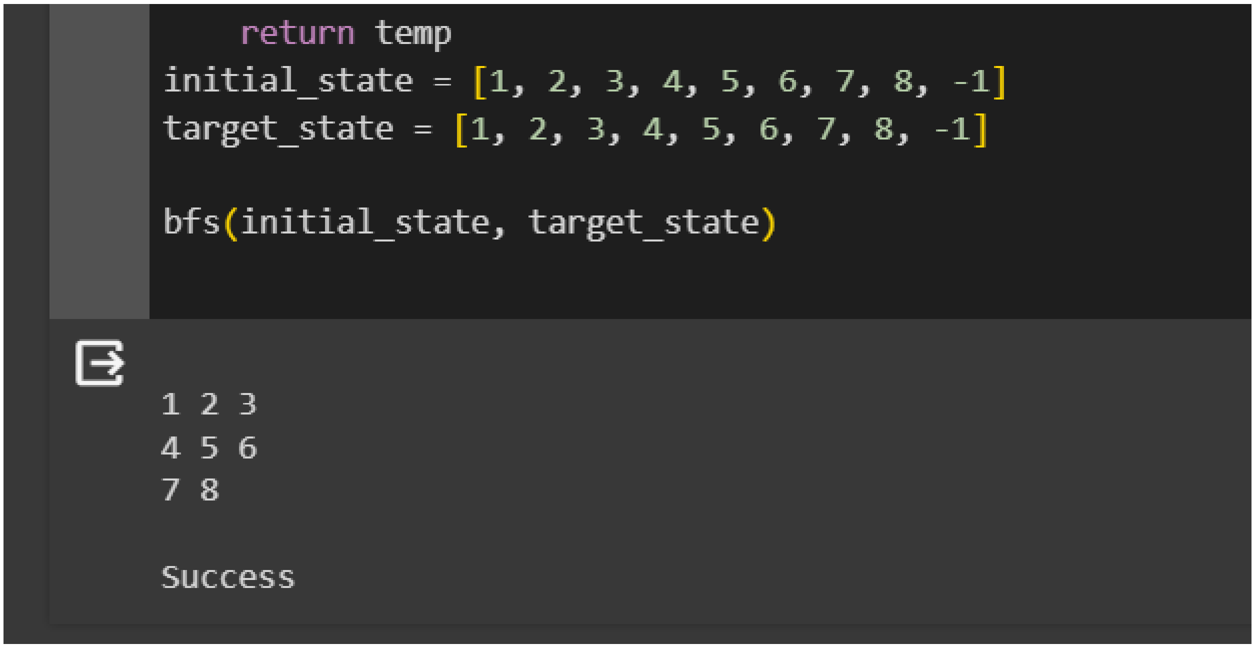
1	2	3
7	4	6
	5	8



Similarly continue branching nodes

```
import numpy as np
  "    Pandas as pd
  "    os

def bfs(src, target):            (BFS code)
    queue = []
    queue append(src)
  exp = []

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

  print(source)

 if source = = target:
     print("success")
    return
  pos_moves to do = []
pos_moves_to_do = possible moves(source

for move in pos_moves_to_do:

    if move not in exp and move
         not in queue:

       queue append(move)
```

```
def possible moves(state, visited_states):
b = state index(0)

d = []                        [ To find
if b not in [0,1,2]:            possible direction
      d append('u')              when the
if b not in [6,7,8]              empty node is
     d append('d')           at diff position ]
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):        [ To move down
                               shift 3 index front
                                & so on ]
      temp = state.copy()

    if m = = 'd':
       temp [b+3], temp [b] = temp[b],
       temp[b-3]
    if m = = 'l':
```



```
    return temp
initial_state = [1, 2, 3, 4, 5, 6, 7, 8, -1]
target_state = [1, 2, 3, 4, 5, 6, 7, 8, -1]

bfs(initial_state, target_state)
```

```
1 2 3
4 5 6
7 8

Success
```