**PROGRAM [2]:**

import copy

from heapq import heappush, heappop

n = 3

rows = [1, 0, -1, 0]

cols = [0, -1, 0, 1]

class PriorityQueue:

def \_\_init\_\_(self):

self.heap = []

def push(self, key):

heappush(self.heap, key)

def pop(self):

return heappop(self.heap)

def empty(self):

return not bool(self.heap)

class Nodes:

def \_\_init\_\_(self, parent, mats, empty\_tile\_posi, costs, levels):

self.parent = parent

self.mats = mats

self.empty\_tile\_posi = empty\_tile\_posi

self.costs = costs

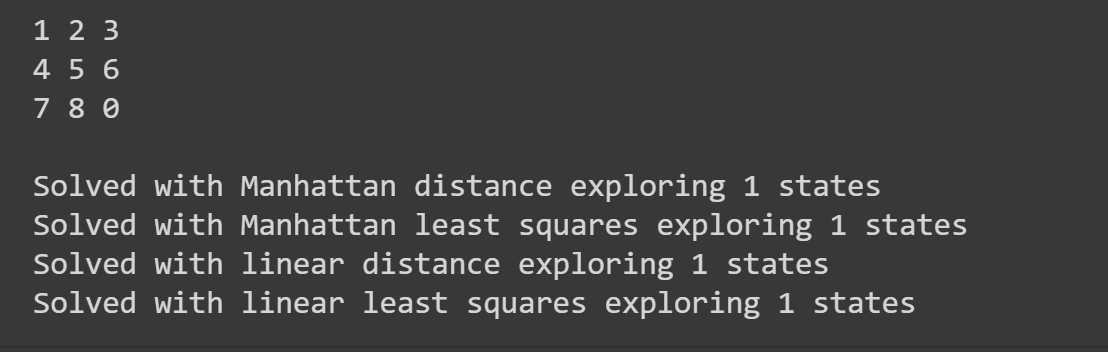
self.levels = levels

def \_\_lt\_\_(self, nxt):

return self.costs < nxt.costs

def calculate\_costs(mats, final):

**OUTPUT [2]:**



return sum(1 for i in range(n) for j in range(n) if mats[i][j] and mats[i][j] != final[i][j])

def new\_nodes(mats, empty\_tile\_posi, new\_empty\_tile\_posi, levels, parent, final):

new\_mats = copy.deepcopy(mats)

x1, y1 = empty\_tile\_posi

x2, y2 = new\_empty\_tile\_posi

new\_mats[x1][y1], new\_mats[x2][y2] = new\_mats[x2][y2], new\_mats[x1][y1]

costs = calculate\_costs(new\_mats, final)

return Nodes(parent, new\_mats, new\_empty\_tile\_posi, costs, levels)

def print\_matrix(mats):

for row in mats:

print(\*row)

print()

def is\_safe(x, y):

return 0 <= x < n and 0 <= y < n

def print\_path(root):

if root is None:

return

print\_path(root.parent)

print\_matrix(root.mats)

def solve(initial, empty\_tile\_posi, final):

pq = PriorityQueue()

costs = calculate\_costs(initial, final)

root = Nodes(None, initial, empty\_tile\_posi, costs, 0)

pq.push(root)

while not pq.empty():

minimum = pq.pop()

if minimum.costs == 0:

print\_path(minimum)

return

for i in range(n):

new\_tile\_posi = [minimum.empty\_tile\_posi[0] + rows[i], minimum.empty\_tile\_posi[1] + cols[i]]

if is\_safe(\*new\_tile\_posi):

child = new\_nodes(minimum.mats, minimum.empty\_tile\_posi, new\_tile\_posi, minimum.levels + 1, minimum, final)

pq.push(child)

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

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

empty\_tile\_posi = [1, 2]

solve(initial, empty\_tile\_posi, final)