PROGRAM

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):

if not self.heap:

return True

else:

return False

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 calculateCosts(mats, final) -> int:

count = 0

for i in range(n):

for j in range(n):

if ((mats[i][j]) and

(mats[i][j] != final[i][j])):

count += 1

return count

def newNodes(mats, empty\_tile\_posi, new\_empty\_tile\_posi,levels, parent, final) -> nodes:

new\_mats = copy.deepcopy(mats)

x1 = empty\_tile\_posi[0]

y1 = empty\_tile\_posi[1]

x2 = new\_empty\_tile\_posi[0]

y2 = new\_empty\_tile\_posi[1]

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

costs = calculateCosts(new\_mats, final)

new\_nodes = nodes(parent, new\_mats, new\_empty\_tile\_posi,costs, levels)

return new\_nodes

def printMatsrix(mats):

for i in range(n):

for j in range(n):

print("%d " % (mats[i][j]), end = " ")

print()

def isSafe(x, y):

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

def printPath(root):

if root == None:

return

printPath(root.parent)

printMatsrix(root.mats)

print()

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

pq = priorityQueue()

costs = calculateCosts(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:

printPath(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 isSafe(new\_tile\_posi[0], new\_tile\_posi[1]):

child = newNodes(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)

OUTPUT

