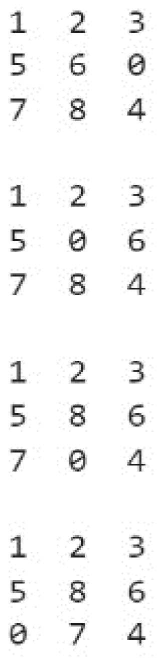
DFS:

```
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:



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

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

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