```
# Importing the heap methods from the python
# library for the Priority Queue
from heapq import heappush, heappop

# This particular var can be changed to transform
# the program from 8 puzzle(n=3) into 15
# puzzle(n=4) and so on ...
n = 3

# bottom, left, top, right
rows = [ 1, 0, -1, 0 ]
cols = [ 0, -1, 0, 1 ]

# creating a class for the Priority Queue
class priorityQueue:

    # Constructor for initializing a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserting a new key 'key'
    def push(self, key):
        heappush(self.heap, key)

    # funct to remove the element that is minimum,
    # from the Priority Queue
    def pop(self):
        return heappop(self.heap)

    # funct to check if the Queue is empty or not
    def empty(self):
        if not self.heap:
            return True
        else:
            return False

# structure of the node
class nodes:
```

```
File  Edit  Shell  Debug  Options  Window  Help
1 1
2 1
3 1
>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
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

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
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

>>>
```

```
    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:

    # Copying data from the parent matrixes to the present matrixes
    new_mats = copy.deepcopy(mats)

    # Moving the tile by 1 position
    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]

    # Setting the no. of misplaced tiles
    costs = calculateCosts(new_mats, final)

    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                      costs, levels)
    return new_nodes

# func to print the N by N matrix
def printMatsrix(mats):

    for i in range(n):
        for j in range(n):
            print("%d " % (mats[i][j]), end = " ")

        print()

# func to know if (x, y) is a valid or invalid
```

```
File  Edit  Shell  Debug  Options  Window  Help
    5  8  6
    0  7  4

>>>
    ========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
    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

>>>
    ========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
    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

>>>
```

```
        # Generating all feasible children
        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]):

                # Creating a child node
                child = newNodes(minimum.mats,
                                 minimum.empty_tile_posi,
                                 new_tile_posi,
                                 minimum.levels + 1,
                                 minimum, final,)

                # Adding the child to the list of live nodes
                pq.push(child)

# Main Code

# Initial configuration
# Value 0 is taken here as an empty space
initial = [ [ 1, 2, 3 ],
            [ 5, 6, 0 ],
            [ 7, 8, 4 ] ]

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
          [ 5, 8, 6 ],
          [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

```
File  Edit  Shell  Debug  Options  Window  Help
    5  8  6
    0  7  4

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
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

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
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
```