⇒ 8-Puzzle problem using A* Algorithm:

1) Node (data, level): initialize node with puzzle
   Start & level

2) def puzzle():
   - accept (): ← Accept start & goal state.
   - f (start, goal) ← calculate [f = h + g]
   - process () ← process A* algorithm

3) Heuristic
   h (start, goal): Manhanlton dist. b/w current & goal state.

4) A* algorithm :-
   1) Initialize start node & add to open list
   2) Loop until & goal state come.
      → Select node have lowest 'f' from open list
      → Display state & check goal
      → generate child node & calculate f
      → And current node to closed list (→ visited) & remove from open list (→ yet to explore)
      → Explore list by 'f'
   3) Display find move

5) · Create puzzle instance
   · call process for the execute.

```
if (mats[i][j] and
    (mats[i][j] != final[i][j]):
    count += 2

return count

def newNodes(mats, empty, new_empty, levels, Parent
             final) -> nodes:

    new_mats = copy.deepcopy(mats)

    x1 = empty_posi[0]
    y1 = empty posi[1]
    x2 = new posi[0]
    y2 = new posi[1]
    new mats[x1][y1], new mats[x1][y1] = new_mats[x2][y2]
                                         new mats[x1][y1]
    costs = calculate cost(new_mats, final)

    new_nodes = nodes(parent, new_mats, new_empty, cost
                      levels)

    return new_nodes

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

child = newNodes(minimum mats)

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

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

#Code

```
import copy
from heapq import heappush, heappop
n=5
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
            False:

class node:
    def __init__(self, parent, mats, empty, level):
            self.parent = parent
            self.mats = mats
            self.empty = empty
            self.costs = costs
            self.level = level

    def calculate costs(mats, final) -> int:
            cost = 0
                for i in range(n):
                    for j in range(n):
```

empty_tile_posi = [1,2]

solve (initial, empty_tile, final)

o/p

1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 6

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4

Enter the start state matrix

1 2 3
4 5 6

Enter the goal state matrix

1 2 3 4
5 6

|
|
\'/

1 2 3 4
5 6
_ 7 8

|
|
\'/

1 2 3 4
5 6
7 _ 8

|
|
\'/

1 2 3
4 5 6
7 8