PRACTICAL – 3

AIM: Write a python program to solve 8 puzzle problem using the A* algorithm.

Code:

```
from enum import Enum
import time
class Action(Enum):
    MoveUp = 1
    MoveDown = 2
    MoveLeft = 3
    MoveRight = 4
    NoAction = 5

class Node:
    def __init__(self, position=[], parent=None,action = Action.NoAction):
        self.position = position
        self.parent = parent
        self.action = action
        if parent:
            self.depth = parent.depth + 1
        else:
            self.depth = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position
    def __lt__(self, other):
         return self.f < other.f
    def __repr__(self):
        strAction = ""
        if(self.action != Action.NoAction):
            strAction = self.action

        return '\n'.join([
        str(strAction),
        str(self.position[:3]),
        str(self.position[3:6]),
        str(self.position[6:9])
        ]).replace('[', '').replace(']', '').replace(',', '').replace('0', '_')
```

```
  def generate_a_star_heuristic_value(self,goal):
    self.h = sum([1 if self.position[i] != goal[i] else 0 for i in range(8)])
    self.f = self.depth + self.h

  def get_all_successors(self):
    successors = []
    i = self.position.index(0)
    if i in [3, 4, 5, 6, 7, 8]:
     new_board = self.position[:]
     new_board[i], new_board[i - 3] = new_board[i - 3], new_board[i]
     successors.append( Node(new_board, self,Action.MoveDown))
    if i in [1, 2, 4, 5, 7, 8]:
     new_board = self.position[:]
     new_board[i], new_board[i - 1] = new_board[i - 1], new_board[i]
     successors.append( Node(new_board, self,Action.MoveRight))
    if i in [0, 1, 3, 4, 6, 7]:
     new_board = self.position[:]
     new_board[i], new_board[i + 1] = new_board[i + 1], new_board[i]
     successors.append( Node(new_board, self,Action.MoveLeft))
    if i in [0, 1, 2, 3, 4, 5]:
     new_board = self.position[:]
     new_board[i], new_board[i + 3] = new_board[i + 3], new_board[i]
     successors.append( Node(new_board, self,Action.MoveUp))
    return successors

class AstarSearch:
 def __init__(self,initial_node, goal_node):
   self.initial_node = Node(initial_node, None)
   self.open = [self.initial_node]
   self.closed = []
   self.goal_node = Node(goal_node, None)
   self.execution()

 def __repr__(self):
  return ""

 def get_path(self,current_node):
  path = []
  while current_node != self.initial_node:
   path.append(current_node)
   current_node = current_node.parent
  path.append(self.initial_node)
  return path[::-1]
```

```
  def perform_algorithm(self):
      while len(self.open) > 0:
          self.open.sort()
          current_node = self.open.pop(0)
          self.closed.append(current_node)
          if current_node == self.goal_node:
              return self.get_path(current_node)
          allSuccesors = current_node.get_all_successors()
          self.add_filtered_successor_to_open(allSuccesors)
      return None

  def add_filtered_successor_to_open(self,allSuccesors):
      for successor in allSuccesors:
        if(successor in self.closed):
          continue
        successor.generate_a_star_heuristic_value(self.goal_node.position)
        if(self.can_add_to_open_list(successor) == True):
          self.open.append(successor)

  def execution(self):
      start_time = time.time()
      path = self.perform_algorithm()
      end_time = time.time()
      self.final_output(path,end_time-start_time)

  def can_add_to_open_list(self,successor):
      for node in self.open:
          if (successor == node and successor.f >= node.f):
              return False
      return True

  def final_output(self,path,execution_time):
    for m in path:
      print(m)
      print()
    print('Total Performed Move: {0}'.format(len(path)-1))
    print("Execution Time= ",round(execution_time*1000, 2),"ms")
    print()

AstarSearch(initial_node=[1, 2, 3, 7, 6, 0, 4, 5, 8],
            goal_node=[1, 2, 3, 4, 5, 6, 7, 8, 0])
```

Output:

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

Action.MoveLeft
1 2 3
4 _ 6
7 5 8

Action.MoveUp
1 2 3
4 5 6
7 _ 8

Action.MoveLeft
1 2 3
4 5 6
7 8 _

Total Performed Move: 3
Execution Time=  0.14 ms
```