from random import randint  
from queue import PriorityQueue  
  
  
class Four\_Puzzle:  
 def \_\_init\_\_(self, size):  
 self.size = size  
 self.puzzle = []  
 self.zero = (0, 0)  
 self.moves = ["U", "D", "L", "R"]  
 count = 1  
 for i in range(0, size):  
 self.puzzle.append([])  
 for j in range(0, size):  
 self.puzzle[i].append(count)  
 count += 1  
 self.puzzle[size - 1][size - 1] = 0  
 self.zero = (size - 1, size - 1)  
  
 def readPuzzle(self, string):  
 a = string.split(" ")  
 count = 0  
 for i in range(0, self.size):  
 for j in range(0, self.size):  
 if int(a[count]) == 0:  
 self.zero = (i, j)  
 self.puzzle[i][j] = int(a[count])  
 count += 1  
  
 def checkPuzzle(self):  
 count = 1  
 for i in range(0, self.size):  
 for j in range(0, self.size):  
 if self.puzzle[i][j] != (count % (self.size \* self.size)):  
 return False  
 count += 1  
 return True  
  
 def swap(self, (x1, y1), (x2, y2)):  
 temp = self.puzzle[x1][y1]  
 self.puzzle[x1][y1] = self.puzzle[x2][y2]  
 self.puzzle[x2][y2] = temp  
  
 def up(self):  
 if self.zero[0] != 0:  
 self.swap((self.zero[0] - 1, self.zero[1]), self.zero)  
 self.zero = (self.zero[0] - 1, self.zero[1])  
  
 def down(self):  
 if self.zero[0] != self.size - 1:  
 self.swap((self.zero[0] + 1, self.zero[1]), self.zero)  
 self.zero = (self.zero[0] + 1, self.zero[1])  
  
 def left(self):  
 if (self.zero[1] != 0):  
 self.swap((self.zero[0], self.zero[1] - 1), self.zero)  
 self.zero = (self.zero[0], self.zero[1] - 1)  
  
 def right(self):  
 if self.zero[1] != self.size - 1:  
 self.swap((self.zero[0], self.zero[1] + 1), self.zero)  
 self.zero = (self.zero[0], self.zero[1] + 1)  
  
 def printPuzzle(self):  
 for i in range(0, self.size):  
 for j in range(0, self.size):  
 print  
 self.puzzle[i][j],  
 print("")  
 # print  
  
 def doMove(self, move):  
 if move == "U":  
 self.up()  
 if move == "D":  
 self.down()  
 if move == "L":  
 self.left()  
 if move == "R":  
 self.right()  
  
 def permute(self, numPerm):  
 for i in range(0, numPerm):  
 self.doMove(self.moves[randint(0, 3)])  
  
 def parseMoveSequence(self, string):  
 for m in string:  
 self.doMove(m)  
  
  
def aSearch(self, heuristic, goal):  
 actual = self.start  
 leaves = PriorityQueue()  
 leaves.put((actual.costHeur(heuristic), actual))  
 closed = list()  
 while True:  
 if leaves.empty():  
 return None  
 actual = leaves.get()[1]  
 if actual.goalState():  
 return actual  
 elif actual.state.puzzle not in closed:  
 closed.append(actual.state.puzzle)  
 succ = actual.succ()  
 while not succ.empty():  
 child = succ.get()  
 leaves.put((child.costHeur(heuristic)+child.depth, child))  
  
  
start = ['2', '3', '1', '0']  
goal = ['1', '2', '0', '3']  
  
obj = aSearch(start, goal)

Mam there is electricity issue in my area .. no laptop charging.ss