Shiva Pooneeth. Kotali
1BM18CS100
AI - Lab1
Shivakotali

```
import numpy as np
from p
from puzzleNode import *
    import copy
    import time
    from queue import PriorityQueue
        from itertools import count

def iterativeDeepeningSearch(startNode):
    maxLayer = 1
    while True:
        dfsList = []
        layer = 0
        dfsList.append((startNode, layer))
        while len(dfsList) != 0:
            top = dfsList.pop()
            tempNode = top[0]
            tempLayer = top[1]
            if tempNode.isGoal():
                traca = []
                ptr = tempNode
                while ptr is not None:
                    traca.append(ptr.nod)
                    ptr = ptr.parent
                return tempLayer, traca
```

```
nextLayer = tmpLayer + 1
if nextLayer > maxLayer:
    continue

validMoves =
tmpNode.getValidMoves()
    for moveChar in validMoves:
        nextNode = copy.deepcopy(tmpNode)
        nextNode.doMove(moveChar)
        if not inDFSNodeList(nextNode, dfsList):

dfsList.append((nextNode, nextLayer))
nextNode.parent = tempNode; maxLayer += 1

iterativeDeepeningSearch()
def inDFSNodeList(tNode, nList):
    for node in nList:
        if (node[0].node == tNode.node).all()
            return True
        return False

test = PuzzleNode()
test.shuffle()
test.show()
step, back = iterativeDeepeningSearch(test)
```

Shiva Kodeli
1BM18CS100

```
Print (step)

while len (bac) != 0;
    n = bac.pop()
    print (n)

import numpy as np
class puzzleNod:
    def __init__(self, init=None):
    self.goal = np.array ([[1,2,3],[4,5,6],[7,8,0]])
    self.nod = self.goal.copy() if init is None
        else init.copy()
            self.io = 2
            self.jo = 2

if init is not None:
  for i in range(3):
  for j in rang (3):
    if self.nodi [i][j] == 0:
        self.io = i
        self.jo = j
        self.parent = None
    def down(self):
    assert self.io > 0
        io = self.io
        jo = self.jo
  self.nodi [io][jo], self.nodi [io-1][jo] = self.nodi [io-1]
                                              [jo], self.nodi [io][j
  self.io -= 1
def up(self):
  assert self.io < 2
    io = self.io
    jo = self.jo
self.nodi [io][jo], self.nodi [io+1][jo] = self.nodi [io+1],
                                    self.nodi [io][jo]
    self.io += 1
```

```
def right(self):
    assert self.jo >= 0
    io = self.io
    jo = self.jo
    self.node[io][jo], self.node[io][jo-1] = self.node[io][jo-1],
                self.node[io][jo]
    self.jo = 0
def left(self):
    assert self.jo < 2
    io = self.io
    jo = self.jo
    self.node[io][jo], self.node[io][jo+1] = self.node[io][jo+1],
                self.node[io][jo]
    self.jo += 1

def getValidMoves(self)
    validDir = []
    if self.io >= 0:
        validDir.append('d')
        if self.io < 2:
            validDir.append('u')
```

```
if self.jo>0:

    validDir.append('r')
    if self.jo < 2:
    validDir.append('l')

    return validDir

def doMove(self, moveChar)

    if move char == 'd':
     self.down()
    elif move char == 'u':
      self.up()
      elif move char == 'r':
        self.righ()
        elif move char == 'l':
          self.left()

def randomStep(self):
    validDir = self.getValidMoves()
    dirNum = len(validDir)
    random Dir = validDir[np.random.randint(0, dirNum)]
          self.doMove(random Dir)
    def shuffle(self, shuffTime = 20):
        for i in range(shuffleTime):
```

Shiva
1BM18CS100

```
self.randomStep()

    def isGoal(self):
        return (self.node == self.goal).all()

    def numOfWrong(self):
        return 9 - np.sum(self.node == self.goal)

    def show(self):
        print(self.node)
```