from copy import deepcopy

class puzzle:

def \_\_init\_\_ (self, starting, parent):

self.board = starting

self.parent = parent

self.f = 0

self.g = 0

self.h = 0

def manhattan(self):

inc = 0

h = 0

for i in range(3):

for j in range(3):

h += abs(inc-self.board[i][j])

inc += 1

return h

def goal(self):

inc = 0

for i in range(3):

for j in range(3):

if self.board[i][j] != inc:

return False

inc += 1

return True

def \_\_eq\_\_(self, other):

return self.board == other.board

def move\_function(curr):

curr = curr.board

for i in range(3):

for j in range(3):

if curr[i][j] == 0:

x, y = i, j

break

q = []

if x-1 >= 0:

b = deepcopy(curr)

b[x][y]=b[x-1][y]

b[x-1][y]=0

succ = puzzle(b, curr)

q.append(succ)

if x+1 < 3:

b = deepcopy(curr)

b[x][y]=b[x+1][y]

b[x+1][y]=0

succ = puzzle(b, curr)

q.append(succ)

if y-1 >= 0:

b = deepcopy(curr)

b[x][y]=b[x][y-1]

b[x][y-1]=0

succ = puzzle(b, curr)

q.append(succ)

if y+1 < 3:

b = deepcopy(curr)

b[x][y]=b[x][y+1]

b[x][y+1]=0

succ = puzzle(b, curr)

q.append(succ)

return q

def best\_fvalue(openList):

f = openList[0].f

index = 0

for i, item in enumerate(openList):

if i == 0:

continue

if(item.f < f):

f = item.f

index = i

return openList[index], index

def AStar(start):

openList = []

closedList = []

openList.append(start)

while openList:

current, index = best\_fvalue(openList)

if current.goal():

return current

openList.pop(index)

closedList.append(current)

X = move\_function(current)

for move in X:

ok = False #checking in closedList

for i, item in enumerate(closedList):

if item == move:

ok = True

break

if not ok: #not in closed list

newG = current.g + 1

present = False

#openList includes move

for j, item in enumerate(openList):

if item == move:

present = True

if newG < openList[j].g:

openList[j].g = newG

openList[j].f = openList[j].g + openList[j].h

openList[j].parent = current

if not present:

move.g = newG

move.h = move.manhattan()

move.f = move.g + move.h

move.parent = current

openList.append(move)

return None

#start = puzzle([[2,3,6],[0,1,8],[4,5,7]], None)

start = puzzle([[5,2,8],[4,1,7],[0,3,6]], None)

# start = puzzle([[0,1,2],[3,4,5],[6,7,8]], None)

#start = puzzle([[1,2,0],[3,4,5],[6,7,8]], None)

result = AStar(start)

noofMoves = 0

if(not result):

print ("No solution")

else:

print(result.board)

t=result.parent

while t:

noofMoves += 1

print(t.board)

t=t.parent

print ("Length: " + str(noofMoves))

OUTPUT:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Length: 18