八数码问题

实验目的

在3×3的棋盘，摆有八个棋子，每个棋子上标有1至8的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始状态转变成目标状态的移动棋子步数最少的移动步骤。

方法

宽度优先搜索

如果搜索是以接近起始节点的程度依次扩展节点的，那么这种搜索就叫做宽度优先搜索。这种搜索是逐层进行的，在对下一层的任一节点进行搜索之前，必须搜索完本层的所有节点。

代码：

\_\_author\_\_ = 'ysc'

import numpy as np

class State:

def \_\_init\_\_(self, state, directionFlag=None, parent=None):

self.state = state

# state is a ndarray with a shape(3,3) to storage the state

self.direction = ['up', 'down', 'right', 'left']

if directionFlag:

self.direction.remove(directionFlag)

# record the possible directions to generate the sub-states

self.parent = parent

self.symbol = ' '

def getDirection(self):

return self.direction

def showInfo(self):

for i in range(3):

for j in range(3):

print(self.state[i, j], end=' ')

print("\n")

print('->')

return

def getEmptyPos(self):

postion = np.where(self.state == self.symbol)

return postion

def generateSubStates(self):

if not self.direction:

return []

subStates = []

boarder = len(self.state) - 1

# the maximum of the x,y

row, col = self.getEmptyPos()

if 'left' in self.direction and col > 0:

#it can move to left

s = self.state.copy()

temp = s.copy()

s[row, col] = s[row, col-1]

s[row, col-1] = temp[row, col]

news = State(s, directionFlag='right', parent=self)

subStates.append(news)

if 'up' in self.direction and row > 0:

#it can move to upper place

s = self.state.copy()

temp = s.copy()

s[row, col] = s[row-1, col]

s[row-1, col] = temp[row, col]

news = State(s, directionFlag='down', parent=self)

subStates.append(news)

if 'down' in self.direction and row < boarder: #it can move to down place

s = self.state.copy()

temp = s.copy()

s[row, col] = s[row+1, col]

s[row+1, col] = temp[row, col]

news = State(s, directionFlag='up', parent=self)

subStates.append(news)

if self.direction.count('right') and col < boarder: #it can move to right place

s = self.state.copy()

temp = s.copy()

s[row, col] = s[row, col+1]

s[row, col+1] = temp[row, col]

news = State(s, directionFlag='left', parent=self)

subStates.append(news)

return subStates

def solve(self):

# generate a empty openTable

openTable = []

# generate a empty closeTable

closeTable = []

# append the origin state to the openTable

openTable.append(self)

steps = 1

# start the loop

while len(openTable) > 0:

n = openTable.pop(0)

closeTable.append(n)

subStates = n.generateSubStates()

path = []

for s in subStates:

if (s.state == s.answer).all():

while s.parent and s.parent != originState:

path.append(s.parent)

s = s.parent

path.reverse()

return path, steps+1

openTable.extend(subStates)

steps += 1

else:

return None, None

if \_\_name\_\_ == '\_\_main\_\_':

# the symbol representing the empty place

# you can change the symbol at here

symbolOfEmpty = ' '

State.symbol = symbolOfEmpty

# set the origin state of the puzzle

originState = State(np.array([[2, 8, 3], [1, 6 , 4], [7, symbolOfEmpty, 5]]))

# and set the right answer in terms of the origin

State.answer = np.array([[1, 2, 3], [8, State.symbol, 4], [7, 6, 5]])

s1 = State(state=originState.state)

path, steps = s1.solve()

if path: # if find the solution

for node in path:

# print the path from the origin to final state

node.showInfo()

print(State.answer)

print("Total steps is %d" % steps)

结果：