**程序报告**

学号： 2113997 姓名：齐明杰

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

八皇后问题： 如何能在 8\*8 的国际象棋棋盘上放置八个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了到达此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

本题采用了回溯算法（Backtracking Algorithm）作为解决八皇后问题的方法。回溯算法是一种常用于解决约束满足问题的算法，其基本思路是通过尝试逐一填充棋盘格子，在每一步填充中，检查当前的填充是否满足约束条件。若满足则继续填充下一行，否则回溯至上一行并尝试其他可能的填充。

在本题中，我们对回溯算法进行了如下改进：

1.使用一维数组表示棋盘，数组的下标表示行，值表示列，这样可以减少内存消耗。

2.对棋盘状态进行深度拷贝，以避免在回溯过程中出现不必要的错误

局限性和常见问题：回溯算法可能会遇到大量重复搜索的问题，因此可以考虑使用记忆化搜索等方法进行优化。

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

import numpy as np # 提供维度数组与矩阵运算

import copy # 从copy模块导入深度拷贝方法

from board import Chessboard

# 基于棋盘类，设计搜索策略

class Game:

def \_\_init\_\_(self, show = True):

"""

初始化游戏状态.

"""

self.chessBoard = Chessboard(show)

self.solves = []

self.gameInit()

# 重置游戏

def gameInit(self, show = True):

"""

重置棋盘.

"""

self.Queen\_setRow = [-1] \* 8

self.chessBoard.boardInit(False)

##############################################################################

#### 请在以下区域中作答(可自由添加自定义函数) ####

#### 输出：self.solves = 八皇后所有序列解的list ####

#### 如:[[0,6,4,7,1,3,5,2],]代表八皇后的一个解为 ####

#### (0,0),(1,6),(2,4),(3,7),(4,1),(5,3),(6,5),(7,2) ####

##############################################################################

# #

def is\_safe(self, board, row, col):

for i in range(row):

if board[i] == col or abs(board[i] - col) == abs(i - row):

return False

return True

def solve\_n\_queens(self, board, row):

if row == len(board):

self.solves.append(copy.deepcopy(board))

print(board)

return

for col in range(len(board)):

if self.is\_safe(board, row, col):

board[row] = col

self.solve\_n\_queens(board, row + 1)

def run(self, row=0):

x = [0]\*8

self.solve\_n\_queens(x, row)

# #

##############################################################################

################# 完成后请记得提交作业 #################

##############################################################################

def showResults(self, result):

"""

结果展示.

"""

self.chessBoard.boardInit(False)

for i,item in enumerate(result):

if item >= 0:

self.chessBoard.setQueen(i,item,False)

self.chessBoard.printChessboard(False)

def get\_results(self):

"""

输出结果(请勿修改此函数).

return: 八皇后的序列解的list.

"""

self.run()

return self.solves

game = Game()

solutions = game.get\_results()

print('There are {} results.'.format(len(solutions)))

game.showResults(solutions[0])

1. **实验结果**

（实验结果，必填）

====================================================================

在实验代码中补充填入代码后点击运行，结果如下：

0 1 2 3 4 5 6 7

0 - - - - - - - -

1 - - - - - - - -

2 - - - - - - - -

3 - - - - - - - -

4 - - - - - - - -

5 - - - - - - - -

6 - - - - - - - -

7 - - - - - - - -

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

There are 92 results.

0 1 2 3 4 5 6 7

0 o - - - - - - -

1 - - - - o - - -

2 - - - - - - - o

3 - - - - - o - -

4 - - o - - - - -

5 - - - - - - o -

6 - o - - - - - -

7 - - - o - - - -

1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

该代码实现了八皇后问题的求解，通过深度优先搜索策略找到了所有可能的解决方案。从输出的结果来看，它成功地找到了所有92个解，并展示了其中一个解的棋盘布局。因此，可以说代码达到了目标预期。

改进的方向：可以尝试使用其他算法，以比较不同算法在求解八皇后问题上的性能差异