**八皇后问题程序报告**

学号： 2112213 姓名：冯思程

1. **问题重述**

首先是本题求解问题的重述：

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。

问题背景：基于python语言进行的逻辑编程实验，将基于python3完成实验，掌握逻辑编程的思想，了解逻辑编程与命令式编程的区别。能够根据规则来进行代码的编写，解决逻辑约束问题。

1. **设计思想**

首先，确定本题对皇后位置对约束有四个：横行无重复、纵行无重复、上斜无重复、下斜无重复。如果用二维矩阵来进行表示和遍历算法会很麻烦，所以考虑用列表来存储皇后位置数据，其中列表的下标就是8个皇后的横行坐标，然后元素的值是纵行坐标，不仅简化了存储，同时也保证了皇后的位置横行无重复这一条件。现在只需要保证皇后位置不在同一个纵行和同一条斜线上，其中通过判断列表中有无重复值和有无坐标连线相比绝对值为1来判断是否存在不符合条件的皇后位置。通过下面的check函数来实现：（如果有不符合条件的皇后位置返回False）

def check(self, list, row):

for i in range(row):

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

return False

return True

然后本题的主要算法是通过回溯算法，通过不断的尝试，遇到错误则回溯，继续尝试，直到所有的可能性都被遍历，得到所有可能的皇后位置序列。如下queen函数：

def queen(self, list, row, n): # list为存储的结果；row是当前行；n为棋盘大小

x=[]

if row == n:#如果都尝试完了，则将此时满足条件的一组答案序列存入solves中

x.extend(list)

self.solves.append(x)

return

for i in range(n): # 这里的i指的是所在列的值，递归进行不断尝试判断，如果不符合条件则继续尝试下一个

list[row] = i

if self.check(list, row):

self.queen(list, row + 1, n)

最后要对run函数进行修改，原函数体是对给的一个解进行运行，将其注释掉，改成要对上文的queen函数进行运行同时将八个皇后和存储的列表信息输入。代码如下：

def run(self, row=0):

n=8

list=[0 for \_ in range(n)]

self.queen(list,row,n)

1. **代码内容**

下面是整体代码展示：

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)

def check(self, list, row):#判断是否有纵行和斜行重复

for i in range(row):

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

return False

return True

def queen(self, list, row, n): # list为存储的结果；row是当前行；n为棋盘大小

x = []

if row == n:

x.extend(list)

self.solves.append(x)

return

for i in range(n): # 这里的i指的是所在列的值

list[row] = i

if self.check(list, row):

self.queen(list, row + 1, n)

def run(self, row=0):

#self.solves.append([0, 6, 4, 7, 1, 3, 5, 2])

n = 8

list = [0 for \_ in range(n)]

self.queen(list, row, n)

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()

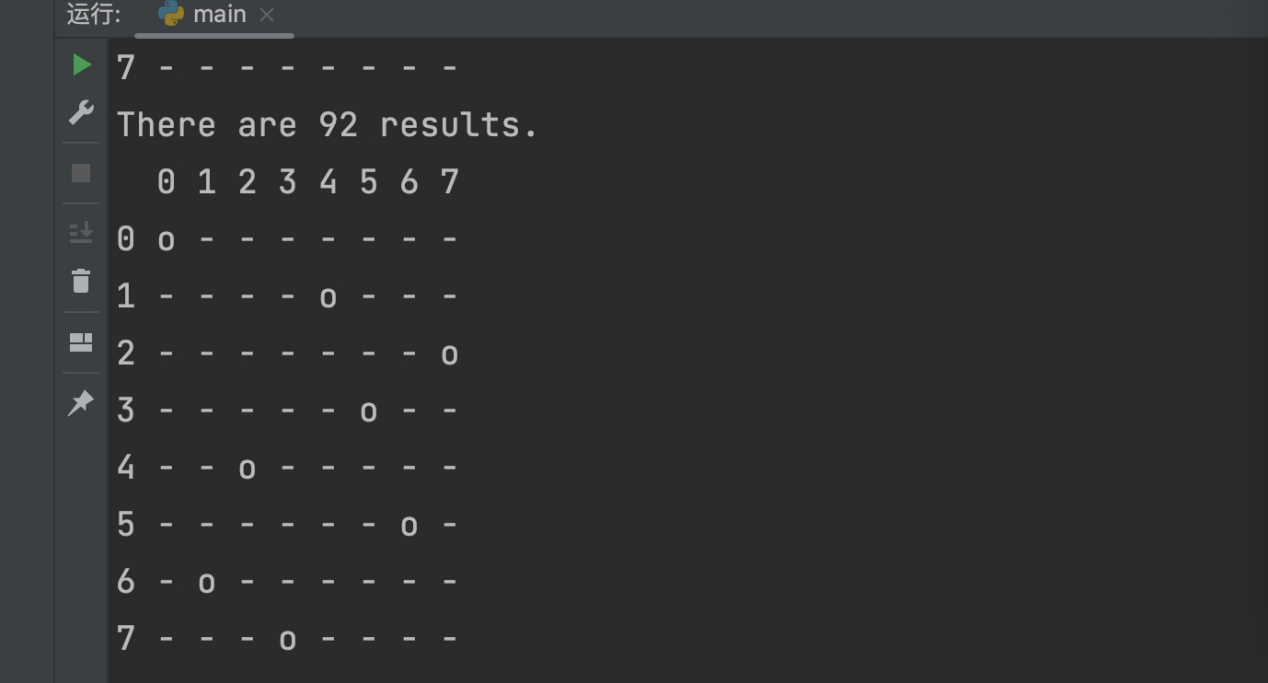
solutions = game.get\_results()

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

game.showResults(solutions[0])

1. **实验结果**

下图为代码在pycharm上的运行结果：（发现可以求出全部的解组合）



1. **总结**

首先，第一个难点是判断出如果进行横纵斜四个方向的条件约束。然后的问题是如果利用回溯算法不断尝试出一个满足条件的皇后位置序列。

考虑能否进行算法优化：本次实验对约束条件的转化是通过判断序列元素有无相同和连线斜率绝对值是否等于1来判断，我们可以利用排列组合的思想，将1至8进行排列组合，则只需考虑不在同一个斜线的条件，进行了一个小的优化。

补充：百度到还有一种算法——遗传算法，将该问题空间模拟成一个生物进化的过程，通过某些操作来淘汰掉适应度函数低的解，逐步接近适应度函数高的较优解，直到得出我们要想要的符合条件的解。