在这个分析报告里，主要进行三个分析：

1. 对于8皇后问题，使用普通遍历法和回溯法，运行时间的对比分析
2. N皇后问题中，N取不同的值，解法的数量的变化
3. 对于回溯法8皇后问题，使用cpu计算和使用cuda计算，运行时间的对比分析
4. **遍历法与回溯法**

除了回溯法（backtrack.py），我还写了8的8次方的遍历法的程序（详情见：ergodic.py）：

其主要的检测函数都差不多，区别是在进行皇后的位置查找方面。遍历法的代码是：

while address[7] != 7 :  
 for a in range(bs):  
 for b in range(bs):  
 for c in range(bs):  
 for d in range(bs):  
 for e in range(bs):  
 for f in range(bs):  
 for g in range(bs):  
 for h in range(bs):  
 address[0] = a  
 address[1] = b  
 address[2] = c  
 address[3] = d  
 address[4] = e  
 address[5] = f  
 address[6] = g  
 address[7] = h  
  
 board = put\_queen(board,address)  
 all\_detect(board, address)

（看起来还挺整齐）

显然，也许遍历法在8皇后问题上可以一战，但是当N取大一些的数时，代码会变得无比冗长。

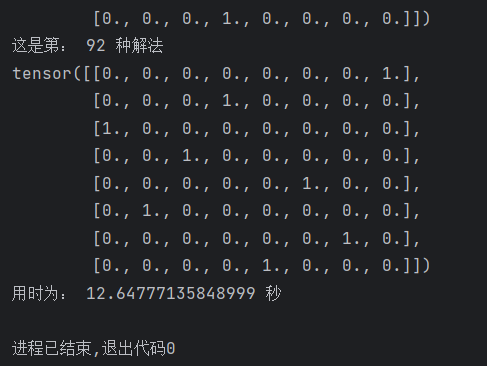
回溯法的代码是：

def put(board:nn.Tensor,i):  
 global num  
 for j in range(bs):  
  
 board[i,j]=1  
 if all\_detect(board) and i<bs-1 :  
 put(board,i+1)  
 if all\_detect(board) and i==bs-1:  
 num+=1  
 print('这是第：',num,'种解法')  
 print(board)  
 board[i,j]=0  
 if j==bs-1 and not all\_detect(board):  
 for k in range(bs):  
 if board[k]==1 and k!=0:  
 board[k-1]=1  
 board[k]=0  
 put(board

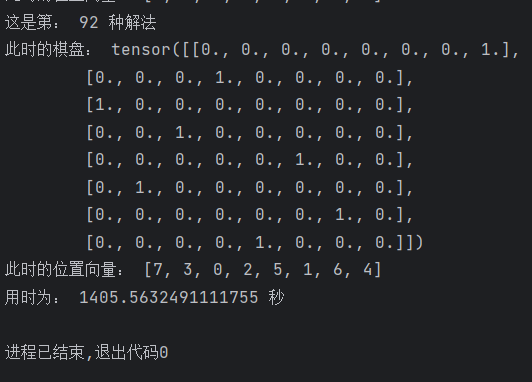
为了计时，导入 time模块，并且在程序开始和结束时，分别定义start\_time和end\_time，最后

print('用时为：',end\_time-start\_time,'秒')

回溯法的运行结果为： 12.65秒



而遍历法的运行结果为：1405.56秒

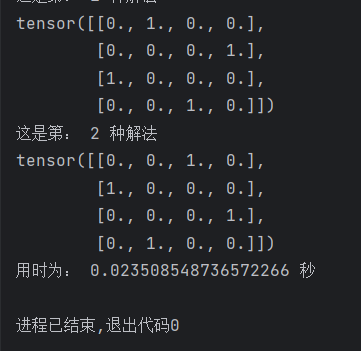


可以推测出，随着N取值增大，两者之间的用时差距会进一步增大

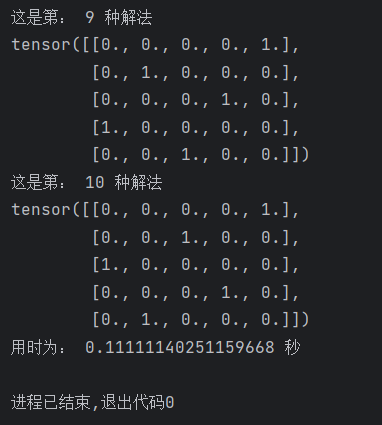
1. **N的不同的值**

通过在backtrack.py中，赋予board\_size不同的值，我们可以观察到：

N = 4 时：有2种

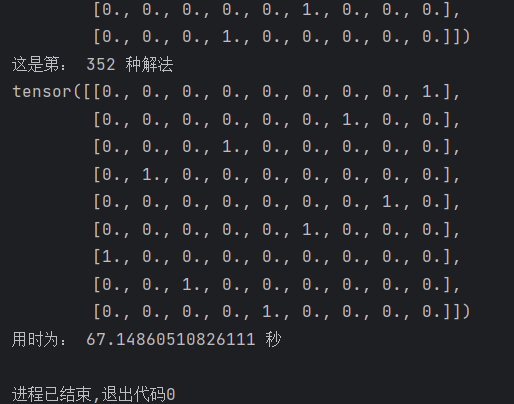


N = 5 时：有10种：

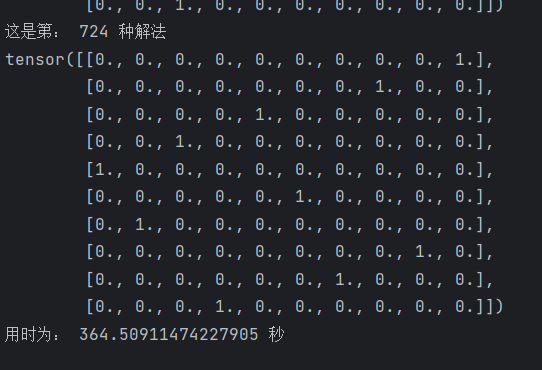


N = 8时有92种

N = 9 时有：352种



N = 10 时，有：724种解法

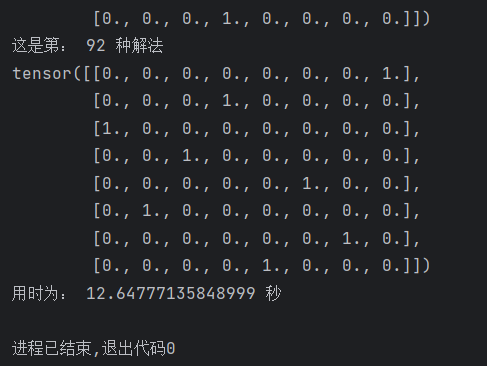


当N增大时，解法也越来越多，而且增量也越来越大。

、

1. **八皇后问题使用cpu与cuda的时间比较**

当使用cpu计算时：用时为12.65秒



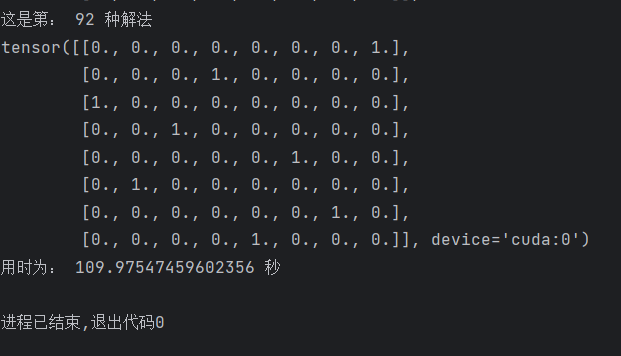
使用cuda计算（显卡是Rtx 3050Ti）:

代码：

device = 'cuda:0'  
board = board.to(device)

运行结果：

用时为109.97秒



可以看出使用cuda运算比使用cpu运算慢了十倍左右，推测原因是因为在进行回溯遍历时，是一个一个进行赋值与修改的，而不是像深度学习网络那样一次性同时进行运算。