NMTTNT_Tuan6_19110413

December 28, 2021

[ ]:
```python
import numpy as np
import random
```

[ ]:
```python
def attacked_queens_pairs(seqs):
    a = np.array([0] * 81)  # Create a one-dimensional array with 81 zeros
    a = a.reshape(9, 9)  # Change to 9 * 9 two-dimensional array. For the␣
↪convenience of later use, only the 8 * 8 parts of the last eight rows and␣
↪columns are used as a blank chessboard
    n = 0  # The number of queens attacking each other is initialized to 0

    for i in range(1, 9):
        if seqs[i-1] != 0: # An element of seqs is 0, which means that no queen␣
↪should be placed in the corresponding chessboard column
            a[seqs[i - 1]][i] = 1  # Generate the corresponding chessboard␣
↪sequence, and place it in the order of the first chessboard column

    for i in range(1, 9):
        if seqs[i - 1] == 0:
            continue # If an element of seqs is 0, it represents the␣
↪corresponding chessboard. If no queen is placed in this column, the next␣
↪column will be judged directly
        for k in list(range(1, i)) + list(range(i + 1, 9)):  # Check whether␣
↪there are other queens on each queen's line
            if a[seqs[i - 1]][k] == 1:  # There are other queens
                n += 1
        t1 = t2 = seqs[i - 1]
        for j in range(i - 1, 0, -1):  # Look at the two diagonals in the left␣
↪half
            if t1 != 1:
                t1 -= 1
                if a[t1][j] == 1:
                    n += 1  # There are other queens on the left half of the␣
↪diagonal

            if t2 != 8:
                t2 += 1
                if a[t2][j] == 1:
```

```
                    n += 1  # There are other queens on the left half of the␣
↪sub diagonal

        t1 = t2 = seqs[i - 1]
        for j in range(i + 1, 9):  # Look at the two diagonals in the right half
            if t1 != 1:
                t1 -= 1
                if a[t1][j] == 1:
                    n += 1  # There are other queens on the right half of the␣
↪diagonal

            if t2 != 8:
                t2 += 1
                if a[t2][j] == 1:
                    n += 1  # There are other queens on the right half of the␣
↪sub diagonal
    return int(n/2)  # Returns n/2, because A attacking B also means B␣
↪attacking A, so returns half of n
```

[ ]:
```
def display_board(seqs):
    """
     Displays the chessboard corresponding to the sequence
    """
    board = np.array([0] * 81)  # Create a one-dimensional array with 81 zeros
    board = board.reshape(9, 9)  # Change to a 9 * 9 two-dimensional array. For␣
↪the convenience of later use, only the 8 * 8 parts of the last eight rows␣
↪and columns are used as a blank chessboard

    for i in range(1, 9):
        board[seqs[i - 1]][i] = 1  # According to the sequence, from the first␣
↪column to the last column, put a queen in the corresponding position to␣
↪generate the chessboard corresponding to the current sequence
    print('The corresponding chessboard is as follows:')
    for i in board[1:]:
        for j in i[1:]:
            print(j, ' ', end="")  # With end, print doesn't wrap
        print()  # After outputting one line, wrap it. This cannot be␣
↪print('\n'), otherwise it will be replaced by two lines
```

[ ]:
```
frontier_priority_queue = [{'pairs':28, 'seqs':[0] * 8}] # The priority queue␣
↪is used to store the unexpanded leaf nodes; the initial state is 8 zeros,␣
↪which means there is no queen on the chessboard; h(n) = the number of queens␣
↪attacking each other, and the initial setting is h(n)=28
solution = []
flag = 0 # The representative has not found a solution
```

```python
while frontier_priority_queue: # If the frontier is not empty, the loop will␣
↪continue. If the solution is found successfully, the loop will output the␣
↪solution. If the frontier is empty, the loop will fail
    first = frontier_priority_queue.pop(0)  # First, the sequence with the␣
↪smallest h(n) is extended; because each sequence is sorted from small to␣
↪large by h(n), the first sequence is extended
    seqs = first['seqs']
    if 0 not in seqs: # Do goal test before extending the node: if there is no␣
↪0 element in the sequence, that is, eight queens have been placed, then the␣
↪sequence is the solution sequence
        solution = seqs
        flag = 1  # success
        break
    nums = list(range(1, 9))  # List with elements 1-8
    for j in range(8): # In the first position of 0 in the sequence, that is,␣
↪the leftmost column without queen, select a row to place queen
        pos = seqs.index(0)
        temp_seqs = list(seqs)
        temp = random.choice(nums)  # Select a random row in the column to␣
↪place the queen
        temp_seqs[pos] = temp # Place the queen on line temp of the column
        nums.remove(temp)  # Remove generated values from nums
        frontier_priority_queue.append({'pairs':
↪attacked_queens_pairs(temp_seqs),'seqs':temp_seqs})
    frontier_priority_queue = sorted(frontier_priority_queue, key=lambda x:
↪x['pairs'])

if solution:
    print('Solution sequence found:' + str(solution))
    display_board(solution)
else:
    print('Algorithm failed, no solution found')
```

```
Solution sequence found:[7, 2, 6, 3, 1, 4, 8, 5]
The corresponding chessboard is as follows:
0  0  0  0  1  0  0  0
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  1
0  0  1  0  0  0  0  0
1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0
```