```python
import random

def random_initial_state(size):
    """Generates a random initial state for the given size, placing one queen in each column."""
    return list(range(size))  # Start with one queen in each column (0 to size-1)

def calculate_heuristic(state):
    """Calculates the number of threatening pairs of queens."""
    threatening_pairs = 0
    size = len(state)
    for i in range(size):
        for j in range(i + 1, size):
            if is_threatening(state[i], i, state[j], j):
                threatening_pairs += 1
    return threatening_pairs

def is_threatening(row1, col1, row2, col2):
    """Checks if two queens threaten each other."""
    return (row1 == row2) or (abs(col1 - col2) == abs(row1 - row2))

def generate_neighbors(state):
    """Generates all possible neighbor states by moving queens to different rows in the same column."""
    neighbors = []
    for col in range(len(state)):
        for row in range(len(state)):
            if row != state[col]:  # Don't move to the same row
                new_state = state.copy()
                new_state[col] = row  # Move queen to new row
                neighbors.append(new_state)
    return neighbors

def hill_climbing(size):
    """Main function to solve the n-Queens problem using hill climbing."""
    current_state = [3, 1, 2, 0]
    current_h = calculate_heuristic(current_state)
    print(current_h)
    while current_h > 0:  # While there are threatening pairs
        neighbors = generate_neighbors(current_state)
        next_state = None
        next_h = current_h

        for neighbor in neighbors:
            neighbor_h = calculate_heuristic(neighbor)
            print(neighbor)
            print(neighbor_h)
            # Check if this neighbor is better
            if neighbor_h < next_h:
                next_state = neighbor
                next_h = neighbor_h

        if next_state is None:  # No better neighbor found
            break

        current_state = next_state
        current_h = next_h

    if current_h == 0:
        return current_state  # Solution found
    else:
        return "No solution found."

# Example usage
size = 4
solution = hill_climbing(size)
print("Solution for 4-Queens:", solution)
```

```
2
[0, 1, 2, 0]
4
[1, 1, 2, 0]
2
[2, 1, 2, 0]
3
[3, 0, 2, 0]
2
[3, 2, 2, 0]
4
[3, 3, 2, 0]
3
[3, 1, 0, 0]
3
[3, 1, 1, 0]
4
[3, 1, 3, 0]
2
[3, 1, 2, 1]
3
[3, 1, 2, 2]
2
[3, 1, 2, 3]
4
Solution for 4-Queens: No solution found.
```

Start coding or generate with AI.

LAB - 5.

Implement Hill Climbing search algorithm to solve N-Queens problem.

```
function hillClimbing8Queens()
    current_state = random_initial_state()
    current_h = calculateHeuristic(current_state).

  while current_h > 0:
      neighbour_state = generateNeighbours(current_state)
     next_state = null.
      next_h = current_h.

  for each neighbour in neighbour_state.
      neighbour_h = calculateHeuristic(neighbour)

    if neighbor_h < next_h :
        next_state = neighbor.
        next_h = neighbor_h

    if next_state is null :
          break;

    current_state = next_state.
    current_h = next_h

    if current_h == 0.
       return current_state.
  else
       return "No solution found within given
               iterations".
```

```
function random_initial_state ():
    return random permutation of
                    [0, 1, 2, 3, 4, 5, 6, 7]

function CalculateHeuristic (state):
    threatening_pairs = 0.
    for i from 0 to 7:
        for j from i+1 to 7:
            if isThreatening (state[i], i, state[j], j)
                Threatening_pair += 1.
    return Threatening_pairs.

function isThreatening (col1, row1, col2, row2):
        return (col1 == col2) or (abs (row1 - row2)
            == abs (col1 - col2))

function generateNeighbors (state)
    neighbour = []
    for row from 0 to 7:
        for col from 0 to 7:
            if col != state (row)
                new_state = state.copy().
                new_state [row] = col
                neighbors.append(new_state)
    return neighbour.
```

State space tree:

Column index representation, index represents column and value represents row

$$\begin{bmatrix} - & - & - & Q \\ - & Q & - & - \\ - & - & Q & - \\ Q & - & - & - \end{bmatrix} \Rightarrow [3\ 1\ 2\ 0]$$

h = 2.
(no. of queen pair attack).

(i). [0, 1, 2, 0] h = 4
(ii). [1, 1, 2, 0] h = 2
(iii). [2, 1, 2, 0] h = 3.
(iv). [3, 0, 2, 0] h = 2.
(v) [3, 2, 2, 0] h = 4
(vi) [3, 3, 2, 0] h = 3
(vii) [3, 1, 0, 0] h = 3.
(viii) [3, 1, 1, 0] h = 4.
(ix) [3, 1, 3, 0] h = 2.
(x) [3, 1, 2, 1] h = 3
(xi) [3, 1, 2, 2] h = 2.
(xii) [3, 1, 2, 3] h = 4.

Solution for 4 queens : No solution found.

28/10/24