N-Queens using Hill Climbing Algorithm:

Date Page

5) N-Queens using Hill Climbing Algorithm

Function N-Queens(problem) returns the state that is local minima:

```
while (True)
    if calculateHeuristics(board) == 0:
        return board
    for each row in board:
        for position in row:
            neighbour = makemove(board, row, position)
            heuristic = calculateHeuristic(neighbour)
            if heuristic < lowestHeuristic:
                bestNeighbour, lowestHeuristic = neighbour, heuristic
    if lowestHeuristic >= calculateHeuristic(board):
        return "local minimum reached", board
    board = bestNeighbour
```

4 Queens Problem:

	0	1	2	3
0				Q
1		Q		
2			Q	
3	Q			

$x_0 = 3$, $x_1 = 1$, $x_2 = 2$, $x_3 = 0$

Neighbours:

$x_0 = 1$, $x_1 = 3$, $x_2 = 2$, $x_3 = 0$, cost = 1

$x_0 = 2$, $x_1 = 1$, $x_2 = 3$, $x_3 = 0$, cost = 1

$x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, cost = 6

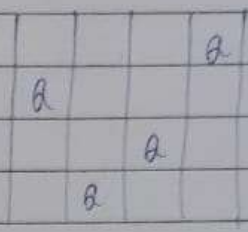
$x_0 = 3$, $x_1 = 2$, $x_2 = 1$, $x_3 = 0$, cost = 6

$x_0 = 3$, $x_1 = 0$, $x_2 = 2$, $x_3 = 1$, cost = 1

Date Page

$x_0 = 3, x_1 = 1, x_2 = 0, x_3 = 2$, cost = 1

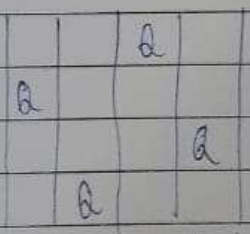
Next state choosen:



Neighbours.

$x_0 = 3, x_1 = 1, x_2 = 2, x_3 = 0$, cost = 2
$x_0 = 2, x_1 = 3, x_2 = 1, x_3 = 0$, cost = 2
$x_0 = 0, x_1 = 3, x_2 = 2, x_3 = 1$, cost = 4
$x_0 = 1, x_1 = 2, x_2 = 3, x_3 = 0$, cost = 4
$x_0 = 1, x_1 = 0, x_2 = 2, x_3 = 3$, cost = 2
$x_0 = 1, x_1 = 3, x_2 = 0, x_3 = 2$, cost = 0

Goal state:



final state

```python
import random

def calculate_attacks(board):
    """Calculate the number of attacking pairs of queens."""
    n = len(board)
    attacks = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j]:
                attacks += 1
            if abs(board[i] - board[j]) == abs(i - j):
                attacks += 1
    return attacks

def generate_neighbors(board):
    """Generate all possible neighbors (configurations) by moving one
queen."""
    neighbors = []
    n = len(board)

    for row in range(n):
        for col in range(n):
            if col != board[row]:
                new_board = board[:]
                new_board[row] = col
                neighbors.append(new_board)

    return neighbors

def hill_climbing(n,board):
    """Solve the N-Queens problem using hill climbing."""

    while True:
        current_attacks = calculate_attacks(board)
        if current_attacks == 0:
            return board
        neighbors = generate_neighbors(board)
        best_neighbor = None
        best_attacks = current_attacks

        for neighbor in neighbors:
            attacks = calculate_attacks(neighbor)
            if attacks < best_attacks:
                best_attacks = attacks
                best_neighbor = neighbor
        if best_attacks >= current_attacks:
            return board

        board = best_neighbor
n = int(input("Enter the size of the board (N): "))
print("Enter the initial configuration of queens (one queen per row):")
initial_board = []
```

```
for i in range(n):
    column = int(input(f"Row {i+1}: Enter the column index for queen (0
to {n-1}): "))
    initial_board.append(column)
solution = hill_climbing(n,initial_board)

print("Final solution:", solution)
for i in range(n):
      for j in range(n):
        if solution[i]==j:
          print("Q",end=" ")
        else:
          print("_",end=" ")
      print()
print("Number of attacks:", calculate_attacks(solution))
print("Name:Vignesh Bhat \nUSN:1BM22CS327")
```

Output:

```
Enter the size of the board (N): 4
Enter the initial configuration of queens (one queen per row):
Row 1: Enter the column index for queen (0 to 3): 3
Row 2: Enter the column index for queen (0 to 3): 3
Row 3: Enter the column index for queen (0 to 3): 3
Row 4: Enter the column index for queen (0 to 3): 3
Final solution: [2, 0, 3, 1]
_ _ Q _
Q _ _ _
_ _ _ Q
_ Q _ _
Number of attacks: 0
Name:Vignesh Bhat
USN:1BM22CS327
```