Experiment No: 6 Date:

8-Queens Problem

Aim: To Implement the 8-Queens Problem using Solution Space Search

Theory:

Introduction:
The 8-Queens Problem is a classic problem in computer science and mathematics where the goal is to place eight queens on a standard 8x8 chessboard in such a way that no two queens threaten each other.

Heuristic-Based Solving:
Heuristic-based solving involves using rules or strategies derived from problem-specific knowledge to guide the search towards promising solutions. In the context of the 8-Queens Problem, heuristic-based solving aims to efficiently explore the solution space by prioritizing moves that are likely to lead to valid solutions.

Approach:

- Initial Chessboard Configuration:
  Start with an initial configuration of the chessboard. This configuration serves as the starting point for the heuristic-based search.
- Heuristic Evaluation Function:
  Define a heuristic evaluation function that quantifies the quality of a solution or a potential move based on problem-specific criteria. The evaluation function should capture properties of the problem such as the number of conflicts between queens.
- Generate Successor States:
  From the initial chessboard configuration, generate successor states by applying heuristic-based moves. These moves are selected based on the heuristic evaluation function and aim to improve the current configuration.
- Evaluate Successor States:
  Evaluate each successor state using the heuristic evaluation function. This evaluation helps prioritize successor states that are likely to lead to better solutions.
- Select Promising Successor States:
  Select a subset of promising successor states based on their heuristic evaluations. These states are considered candidates for further exploration.
- Iterative Improvement:
  Iterate the process by generating successor states from the selected candidates and evaluating them. Continue selecting and exploring promising states until a satisfactory solution is found.
- Termination and Output:
  Terminate the search when a valid solution is found. Output the best solution found during the search process.

Algorithm:

1. Initialize current_board with initial_board.
2. current_heuristic = Heuristic(current_board)
3. same_state_count = 0
4. max_same_state_count = 3
5. Print "Initial Board:", current_board
6. Print "Initial Heuristic:", current_heuristic
7. while current_heuristic > 0 do:
8. row1, row2 = RandomSample(1, 8, 2) // Generate random move (swap two rows)
9. ApplyMove(current_board, row1, row2) // Apply the move to generate a new board configuration
10. new_heuristic = Heuristic(current_board) // Calculate the new heuristic value
11. if new_heuristic < current_heuristic then:
12. current_heuristic = new_heuristic // Update current heuristic
13. same_state_count = 0 // Reset same state counter
14. Print "Current Board:", current_board
15. Print "Current Heuristic:", current_heuristic
16. else:
17. UndoMove(current_board, row1, row2) // Undo the move if it doesn't lead to improvement
18. same_state_count = same_state_count + 1
19. if same_state_count >= max_same_state_count then: //if same moves gets repeated
20. row1, row2 = RandomSample(1, 8, 2) // Perform a random move to a neighboring state
21. ApplyMove(current_board, row1, row2)
22. same_state_count = 0 // Reset same state counter
23. Print "Solution Found!"
24. Print "Final Board:", current_board
25. Print "Final Heuristic:", current_heuristic

Example:

Imagine you're solving the 8-Queens Problem using a heuristic-based local search. You start with an initial chessboard configuration where queens are randomly placed on the board. Your goal is to move the queens around until no two queens threaten each other. At each step, you evaluate the current configuration using a heuristic function, which measures the number of conflicts between queens. You then make random moves, swapping the positions of two queens, to try and reduce the number of conflicts. If a move results in fewer conflicts, you accept it and continue exploring. But if the move doesn't improve the configuration, you might still accept it occasionally to avoid getting stuck in local minima. After several iterations, you find a configuration with no conflicts, indicating a solution to the problem.

Initial Board: [1, 4, 6, 8, 2, 3, 5, 7]
Initial Heuristic: 3
Current Board: [1, 4, 2, 8, 6, 3, 5, 7]
Current Heuristic: 2
Current Board: [2, 4, 1, 8, 5, 3, 6, 7]
Current Heuristic: 1
Current Board: [3, 6, 4, 1, 8, 5, 7, 2]
Current Heuristic: 0
Solution Found!
Final Board: [3, 6, 4, 1, 8, 5, 7, 2]
Final Heuristic: 0

Program:

```
import random
def heuristic(board):
    conflicts = 0
    for i in range(len(board)):
        for j in range(i+1, len(board)):
            if board[i] == board[j] or abs(i - j) == abs(board[i] - board[j]):
                conflicts += 1
    return conflicts

def solve_queens(initial_board):
    current_board = initial_board.copy()
    current_heuristic = heuristic(current_board)
    same_state_count = 0
    max_same_state_count = 3
    print("Initial Board:", current_board)
    print("Initial Heuristic:", current_heuristic)

    while current_heuristic > 0:
        # Generate a random move (swap two rows)
        row1, row2 = random.sample(range(1, 9), 2)
        current_board[row1-1], current_board[row2-1] = current_board[row2-1], current_board[row1-1]
        new_heuristic = heuristic(current_board)
        if new_heuristic < current_heuristic:
            current_heuristic = new_heuristic
            same_state_count = 0  # Reset same state counter
            print("Current Board:", current_board)
            print("Current Heuristic:", current_heuristic)
        else:
            # Undo the move if it doesn't lead to improvement
          current_board[row1-1], current_board[row2-1] = current_board[row2-1], current_board[row1-1]
            same_state_count += 1
       #to solve local minima
        if same_state_count >= max_same_state_count:
            # Perform a random move to a neighboring state
            row1, row2 = random.sample(range(1, 9), 2)
          current_board[row1-1], current_board[row2-1] = current_board[row2-1], current_board[row1-1]
            same_state_count = 0  # Reset same state counter

    print("Solution Found!")
    print("Final Board:", current_board)
    print("Final Heuristic:", current_heuristic)

initial_board = []
t=8
print("Initial Board")
while t:
    inp=int(input())  # column is input and row is the index
    initial_board.append(inp)
    t=t-1
solve_queens(initial_board)
```

Output

```
Initial Board
1
2
3
4
5
6
7
8
Initial Board: [1, 2, 3, 4, 5, 6, 7, 8]
Initial Heuristic: 28
Current Board: [5, 2, 3, 4, 1, 6, 7, 8]
Current Heuristic: 18
Current Board: [5, 2, 3, 4, 1, 6, 8, 7]
Current Heuristic: 10
Current Board: [5, 2, 3, 4, 6, 1, 8, 7]
Current Heuristic: 6
Current Board: [8, 2, 3, 1, 6, 4, 5, 7]
Current Heuristic: 2
Current Board: [8, 5, 3, 1, 6, 4, 2, 7]
Current Heuristic: 1
Current Board: [5, 8, 4, 1, 3, 6, 2, 7]
Current Heuristic: 0
Solution Found!
Final Board: [5, 8, 4, 1, 3, 6, 2, 7]
Final Heuristic: 0
```

```
Initial Board
1
4
6
8
2
3
5
7
Initial Board: [1, 4, 6, 8, 2, 3, 5, 7]
Initial Heuristic: 3
Current Board: [1, 4, 2, 8, 6, 3, 5, 7]
Current Heuristic: 2
Current Board: [2, 4, 1, 8, 5, 3, 6, 7]
Current Heuristic: 1
Current Board: [3, 6, 4, 1, 8, 5, 7, 2]
Current Heuristic: 0
Solution Found!
Final Board: [3, 6, 4, 1, 8, 5, 7, 2]
Final Heuristic: 0
```

Conclusion: Solved 8-queens problem using Solution Space Search with successful execution of programs.