**EXPERIMENT NO : 06 DATE : 18/03/24**

**Aim**: To implement the N-Queens problem using solution space search.

**Theory**:

We have N queens and an NxN Chess board having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any other queen placed on same row, or column or diagonal. We have to find the proper placement of queens on the Chess board in such a way that no queen attacks other queen” .In the game of chess, the queen is a powerful piece. It can attack by moving any number of spaces in its current row, in its column or diagonally. In the N- queens puzzle, N queens must be placed on a standard N×N chess board so that no queen can attack another. The center figure below shows an invalid solution; two queens can attack each other diagonally. The figure on the right shows a valid solution. Given a description of a chess board.



**Constraint Satisfaction Problem:**

Next, Q3 cannot be placed in position (3, 1) as Q1 attacks her. And it cannot be placed at (3, 2), (3, 3) or (3, 4) as Q2 attacks her. There is no way to put Q3 in third row. Hence, the algorithm backtracks and goes back to the previous solution and readjusts the position of queen Q2. Q2 is moved from positions (2, 3) to (2, 4). Partial solution is Now, Q3 can be placed at position (3, 2).Partial solution is . Queen Q4 cannot be placed anywhere in row four. So again, backtrack to the previous solution and readjust the position of Q3. Q3 cannot be placed on (3, 3) or(3, 4). So, the algorithm backtracks even further. All possible choices for Q2 are already explored, hence the algorithm goes back to partial solution and moves the queen Q1 from (1, 1) to (1,2) and this process continues until a solution is found.

**Problem Definition**

Given 4 x 4 chessboard, arrange four queens in a way, such that no two queens attack each other. That is, no two queens are placed in the same row, column, or diagonal.



We have to arrange four queens, Q1, Q2, Q3 and Q4 in 4 x 4 chessboard. We will put ith queen in ith row. Let us start with position (1, 1). Q1 is the only queen, so there is no issue. partial solution is We cannot place Q2 at positions (2, 1) or (2, 2). Position (2, 3)is acceptable. Partial solution is . Next, Q3 cannot be placed in position (3, 1) as Q1 attacks her. And it cannot be placed at (3, 2), (3, 3) or (3, 4) as Q2 attacks her. There is no way to put Q3 in third row. Hence, the algorithm backtracks and goes back to the previous solution and readjusts the position of queen Q2. Q2 is moved from positions (2, 3) to (2, 4). Partial solution is . Now, Q3 can be placed at position (3, 2). Partial solution is . Queen Q4 cannot be placed anywhere in row four. So again, backtrack to the previous solution and readjust the position of Q3. Q3 cannot be placed on (3, 3) or(3, 4). So, the algorithm backtracks even further. All possible choices for Q2 are already explored, hence the algorithm goes back to partial solution and moves the queen Q1 from (1, 1) to (1, 2). And this process continues until a solution is found.

Rules Of 8 Queens Problem

If the queen is at row r and column c, then it can attack any square (r', c') such that:

• r' = r (horizontal movement)

• c' = c (vertical movement)

• r'+c' = r+c (northwest-southeast movement)

• r'-c' = r-c (northeast-southwest movement)

**Code:**

import random

def generate\_random\_state():

return [random.randint(0, 7) for \_ in range(8)]

def conflicts(state, row, col):

# count the number of conflicts

count = 0

for i in range(len(state)):

if i != row: # makes sure we are not checking conflicts int he same row

if state[i] == col or abs(i - row) == abs(

state[i] - col

): # horiz, vert, diag

count += 1

return count

def heuristic(state):

# number of conflicts in the current state

total\_conflicts = 0

for i in range(len(state)):

total\_conflicts += conflicts(state, i, state[i])

return total\_conflicts

def print\_board(state):

for i in range(8):

for j in range(8):

if state[i] == j:

print("Q ", end="")

else:

print(". ", end="")

print()

print()

def min\_conflicts(state, max\_iter=10):

for \_ in range(max\_iter):

if heuristic(state) == 0: # no conflicts

return state

row = min\_conflict\_row(state)

col = min\_conflict\_col(state, row)

state[row] = col

print\_board(state)

return None

def min\_conflict\_row(state):

min\_conflicts = float("inf")

min\_row = -1

for i in range(8):

for j in range(8):

if j != state[i]:

state\_copy = state[:]

state\_copy[i] = j

conflicts\_count = heuristic(state\_copy)

if conflicts\_count < min\_conflicts:

min\_conflicts = conflicts\_count

min\_row = i

return min\_row

def min\_conflict\_col(state, row):

#Find the column with the minimum conflicts for a queen in a given row."""

min\_conflicts = float("inf")

min\_col = -1

for col in range(8):

conflict\_count = conflicts(state, row, col)

if conflict\_count < min\_conflicts:

min\_conflicts = conflict\_count

min\_col = col

return min\_col

# MAIN

initial\_state = generate\_random\_state()

# initial\_state = [0, 1, 2, 3, 4, 5, 6, 7]

for x in initial\_state:

print(x + 1, end=" ")

print()

print("Initial State")

print\_board(initial\_state)

solution = min\_conflicts(initial\_state)

# Print the solution

if solution:

print("Solution found:")

print\_board(solution)

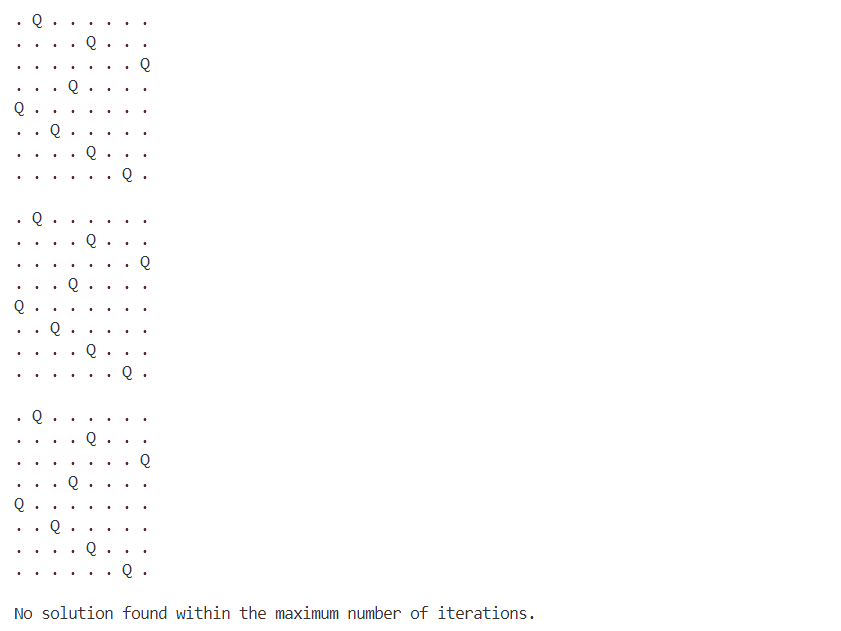
else:

print("No solution found within the maximum number of iterations.")

**Output:**

****

****

****

**Conclusion:**

Implementation of the N Queens Problem Using Solution Space Search assuming a solution space was carried out by tracing the algorithm and above output was obtained during the same