Experiment:-2

Objective:- Implement Basic Search Strategies 8-queens Problem.

Theory:-

The **8-Queens problem** is a classic **constraint satisfaction problem (CSP)** in artificial intelligence and computer science. It involves placing **8 queens** on a standard **8×8 chessboard** such that **no two queens attack each other**. This means:

- No two queens share the same **row**
- No two queens share the same **diagonal**

Algorithm:-

Step by step approach:

1. Initialize an empty 8×8 chessboard with all positions set to 0.
2. Start with the first row (row 0) and try placing a queen in each column. 3. For each placement, check if the position is safe (not attacked by any previously placed queen). A position is unsafe if another queen is in the same column or on the same diagonal. 4. If a safe position is found, place the queen (set position to 1) and recursively try to place queens in subsequent rows. Otherwise, backtrack by removing the queen and trying the next column.
5. If all rows are successfully filled (8 queens placed), a valid solution is

found. **Program:-**

```
def printSolution(board):
 """Print the chessboard configuration."""
 for row in board:
 print(" ".join("Q" if col else "." for col in row))
 print("\n")

def isSafe(board, row, col, n):
 """Check if placing a queen at board[row][col] is safe."""
 # Check column



 for i in range(row):
 if board[i][col]:
 return False

 # Check upper-left diagonal
 i, j = row, col
 while i >= 0 and j >= 0:
```

```
if board[i][j]:
return False
i -= 1

    j -= 1
# Check upper-right diagonal
i, j = row, col
while i >= 0 and j < n:
if board[i][j]:
return False
i -= 1
j += 1

return True

def solveNQueens(board, row, n):
"""Use backtracking to solve the N-Queens problem."""
if row == n:
printSolution(board)
return True

result = False
for col in range(n):
if isSafe(board, row, col, n):
# Place the queen
board[row][col] = 1
# Recur to place the rest of the queens
result = solveNQueens(board, row + 1, n) or result
# Backtrack
board[row][col] = 0

return result

def nQueens(n):
"""Driver function to solve the N-Queens problem."""
board = [[0] * n for _ in range(n)]
if not solveNQueens(board, 0, n):
print("No solution exists.")
else:
print("Solutions printed above.")

# Solve the 8-Queens problem
nQueens(8)
```

Output:-

Q . . . . . .
. . . . Q . . .
. . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . Q .
. Q . . . . . .
. . . Q . . . .

Q . . . . . .
. . . . . Q . .
. . . . Q
. . Q . . . . .
. . . . . Q .
. . . Q . . . .
. Q . . . . . .
. . . . Q . . .

. . . . Q
. . . Q . . . .
Q . . . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . Q .
. . . . Q . . .