**Lab Assignment 6: Implementation of Basic Search Strategies for the 8-Queens Problem**

**1. Objective:**

The objective of this lab assignment is to implement basic search strategies to solve the **8-Queens problem**. The goal is to place eight queens on an 8x8 chessboard such that no two queens threaten each other. This means that no two queens can share the same row, column, or diagonal.

**2. Problem Statement:**

The 8-Queens problem is a classic combinatorial problem in which the challenge is to find all possible arrangements of 8 queens on a chessboard. Each solution must ensure that no two queens are placed in such a way that they can attack each other.

**3. Theory:**

**3.1. Backtracking Approach:**

The most common approach to solve the N-Queens problem, including the 8-Queens problem, is through **backtracking**. This technique involves placing queens on the board one row at a time and checking for conflicts with previously placed queens. If a conflict occurs, the algorithm backtracks to try a different position.

**3.2. Constraints:**

* **Row Constraint**: Each queen must be placed in a different row.
* **Column Constraint**: Each queen must be placed in a different column.
* **Diagonal Constraint**: No two queens can be on the same diagonal, which can be checked using the absolute difference between the row and column indices.

**3.3. Search Strategies:**

We can implement various search strategies to solve the 8-Queens problem:

1. **Depth-First Search (DFS)**: Recursively place queens row by row.
2. **Backtracking**: Utilize DFS with backtracking to eliminate invalid configurations.

**4. Algorithm Design:**

**4.1. Backtracking Algorithm Steps:**

1. Start placing queens in the first row.
2. For each column in the current row:
   * Check if placing the queen at (row, column) is safe.
   * If it is safe, place the queen and move to the next row.
   * If placing the queen leads to a solution, record the solution.
   * If not, backtrack by removing the queen and trying the next column.
3. Repeat until all rows are filled or all possibilities are exhausted.

**4.2. Pseudocode:**

Here is the pseudocode for solving the 8-Queens problem using the backtracking approach:

plaintext

Copy code

function solveNQueens(N):

board = initialize(N)

solutions = []

placeQueens(board, 0, solutions)

return solutions

function placeQueens(board, row, solutions):

if row == N: // All queens are placed

solutions.append(clone(board))

return

for col in range(0, N):

if isSafe(board, row, col):

board[row][col] = 'Q' // Place queen

placeQueens(board, row + 1, solutions) // Recur to place next queen

board[row][col] = '.' // Backtrack

function isSafe(board, row, col):

for i in range(0, row): // Check column

if board[i][col] == 'Q':

return false

for i, j in zip(range(row-1, -1, -1), range(col-1, -1, -1)): // Check upper left diagonal

if board[i][j] == 'Q':

return false

for i, j in zip(range(row-1, -1, -1), range(col+1, N)): // Check upper right diagonal

if board[i][j] == 'Q':

return false

return true

**5. Implementation:**

Here is a sample Python implementation of the above pseudocode:

python

Copy code

def solve\_n\_queens(N):

board = [['.' for \_ in range(N)] for \_ in range(N)]

solutions = []

place\_queens(board, 0, solutions)

return solutions

def place\_queens(board, row, solutions):

if row == len(board): # All queens are placed

solutions.append([''.join(r) for r in board])

return

for col in range(len(board)):

if is\_safe(board, row, col):

board[row][col] = 'Q' # Place queen

place\_queens(board, row + 1, solutions) # Recur to place next queen

board[row][col] = '.' # Backtrack

def is\_safe(board, row, col):

# Check the column for another queen

for i in range(row):

if board[i][col] == 'Q':

return False

# Check upper left diagonal

for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):

if board[i][j] == 'Q':

return False

# Check upper right diagonal

for i, j in zip(range(row - 1, -1, -1), range(col + 1, len(board))):

if board[i][j] == 'Q':

return False

return True

# Example usage:

N = 8

solutions = solve\_n\_queens(N)

for solution in solutions:

for row in solution:

print(row)

print()

**6. Expected Output:**

When you run the implementation for N = 8, you will receive multiple solutions, each representing a valid arrangement of 8 queens on the chessboard. Each solution will be printed with Q denoting a queen and . representing an empty square.

Example output for one solution:

css

Copy code

Q.......

..Q.....

....Q...

......Q.

......Q.

...Q....

.....Q..

.....Q..

**7. Conclusion:**

The implementation demonstrates the use of backtracking as a basic search strategy to solve the 8-Queens problem. By exploring all possible configurations and eliminating invalid moves, the algorithm effectively finds all valid arrangements for the queens on the chessboard.

**8. References:**

* Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
* Knuth, D. E. (1971). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.