# Assignment 6: Implement Basic Search Strategies – 8 Queen Problem

## Aim:

To implement basic search strategies by solving the 8-Queen problem using backtracking.

## Theory:

The 8-Queen problem is a classic example of a constraint satisfaction and backtracking problem. The objective is to place eight queens on an 8×8 chessboard such that no two queens attack each other. A queen can attack any piece that is on the same row, column, or diagonal. To solve this, we use a basic search strategy known as backtracking.  
  
Backtracking is a form of depth-first search where we try to build a solution incrementally, and if we encounter a situation that violates the problem constraints, we backtrack to the previous step and try another possibility. This process continues until a valid solution is found or all possibilities are exhausted.

## Algorithm / Steps:

* 1. Start with an empty 8×8 chessboard.
* 2. Place a queen in the first column and check if it can be placed safely.
* 3. Move to the next column and repeat the process for all rows.
* 4. If a safe position is found, place the queen and move to the next column.
* 5. If no safe position is available, backtrack to the previous column and move the queen to a new position.
* 6. Continue this process until all queens are placed safely on the board or all possibilities are exhausted.

## Code:

*#include <iostream>  
#include <vector>  
  
using namespace std;  
  
// Function to print the chessboard  
void printBoard(const vector<vector<int>>& board) {  
 for (const auto& row : board) {  
 for (int cell : row) {  
 cout << (cell == 1 ? "Q " : ". ");  
 }  
 cout << endl;  
 }  
}  
  
// Function to check if a position is safe for a queen  
bool isSafe(const vector<vector<int>>& board, int row, int col, int N) {  
 for (int i = 0; i < col; ++i)  
 if (board[row][i] == 1)  
 return false;  
  
 for (int i = row, j = col; i >= 0 && j >= 0; --i, --j)  
 if (board[i][j] == 1)  
 return false;  
  
 for (int i = row, j = col; i < N && j >= 0; ++i, --j)  
 if (board[i][j] == 1)  
 return false;  
  
 return true;  
}  
  
// Recursive function to solve the 8-Queens problem  
bool solveNQueensUtil(vector<vector<int>>& board, int col, int N) {  
 if (col >= N)  
 return true;  
  
 for (int i = 0; i < N; ++i) {  
 if (isSafe(board, i, col, N)) {  
 board[i][col] = 1;  
  
 if (solveNQueensUtil(board, col + 1, N))  
 return true;  
  
 board[i][col] = 0;  
 }  
 }  
 return false;  
}  
  
int main() {  
 int N = 8;  
 vector<vector<int>> board(N, vector<int>(N, 0));  
  
 if (solveNQueensUtil(board, 0, N)) {  
 cout << "Solution found:\n";  
 printBoard(board);  
 } else {  
 cout << "Solution does not exist.\n";  
 }  
 return 0;  
}*

## Output:

*Solution found:  
Q . . . . . . .  
. . . . Q . . .  
. . . . . . . Q  
. . . . . Q . .  
. . Q . . . . .  
. . . . . . Q .  
. Q . . . . . .  
. . . Q . . . .*

## Conclusion:

The 8-Queen problem was successfully implemented using the backtracking approach. This assignment demonstrates how basic search strategies can be applied to constraint satisfaction problems, where backtracking helps explore all possible configurations efficiently until a valid solution is found.