Experiment No. 5

Title: To Solve the N-Queens Problem using Backtracking in Python.

Aim:
To implement the N-Queens problem using Python, understanding the concept of backtracking, and developing a solution to place N chess queens on an N×N chessboard so that no two queens threaten each other.

Objectives:

1. Develop a Python program to solve the N-Queens problem.
2. Understand and implement the backtracking algorithm for solving constraint satisfaction problems.

Problem Statement:

Given an N×N chessboard, the N-Queens problem requires placing N queens on the board such that no two queens threaten each other. In chess, a queen can attack any piece in the same row, column, or diagonal. The task is to find all possible configurations of placing N queens on the board where no two queens can attack each other.

Software/s: Python (version 3.x)

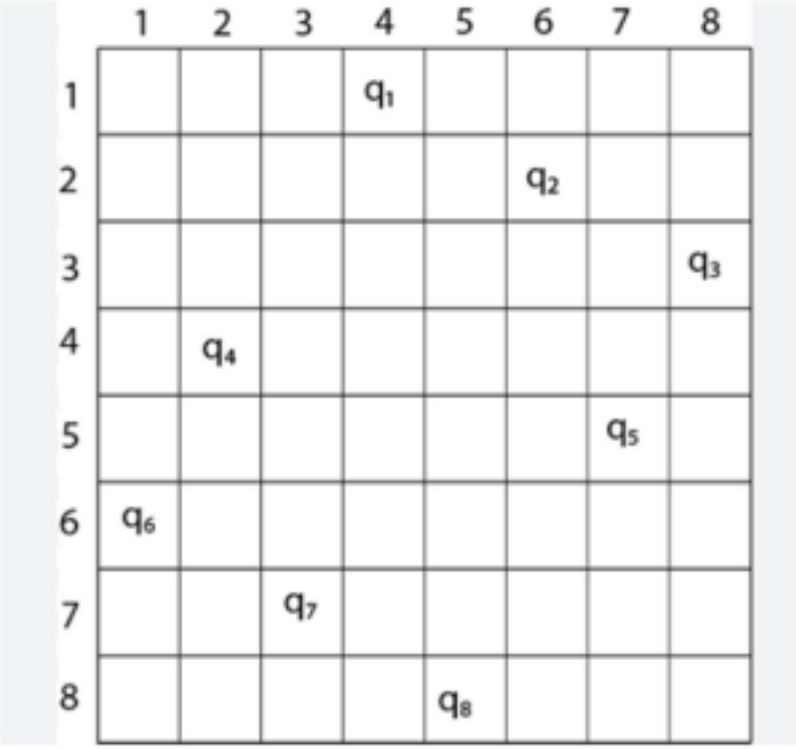
Theory:

The N-Queens problem is a classical problem in computer science and combinatorial optimization. It can be solved using various techniques, with backtracking being one of the most efficient methods. Backtracking is a systematic way of exploring all possible solutions by incrementally building candidates for a solution and discarding those that fail to satisfy the constraints.
Backtracking is a recursive approach for solving any problem where we must search among all the possible solutions following some constraints. More

precisely, we can say that it is an improvement over the brute force technique. In this blog, we will learn one popular DSA question: 8 queen's problem using Backtracking.

Similar to the N-Queens problem, the 8-Queen problem can be efficiently solved using backtracking. Backtracking is a systematic algorithmic technique that incrementally builds candidates for a solution and discards those that fail to satisfy the problem's constraints. In the context of the 8-Queen problem, backtracking involves exploring different configurations of queen placements on the chessboard while ensuring that no two queens share the same row, column, or diagonal.

Due to the larger problem size, the search space increases exponentially, necessitating an efficient algorithm to find solutions within a reasonable time frame. Backtracking offers a practical solution approach by pruning branches of the search tree whenever a partial candidate solution cannot be completed to a valid solution. This reduces the number of candidate solutions that need to be explored, leading to an optimized search process.



Algorithm:

Let's go through the steps below to understand how this algorithm of solving the 8 queens problem using backtracking works:

- Step 1: Traverse all the rows in one column at a time and try to place the queen in that position.

- Step 2: After coming to a new square in the left column, traverse to its left horizontal direction to see if any queen is already placed in that row or not. If a queen is found, then move to other rows to search for a possible position for the queen.
- Step 3: Like step 2, check the upper and lower left diagonals. We do not check the right side because it's impossible to find a queen on that side of the board yet.
- Step 4: If the process succeeds, i.e. a queen is not found, mark the position as '1' and move ahead.
- Step 5: Recursively use the above-listed steps to reach the last column. Print the solution matrix if a queen is successfully placed in the last column.
- Step 6: Backtrack to find other solutions after printing one possible

solution. Pseudocode:

```
procedure solve_8_queens(board, row):
 if row == 8:
 return true // All queens are placed successfully

 for col from 0 to 7:
 if is_safe(board, row, col):
 board[row] = col // Place queen in the current row and column
 if solve_8_queens(board, row + 1):
 return true // Solution found
 board[row] = -1 // Backtrack: Remove queen from the current position

return false // No safe position found for the current row

function is_safe(board, row, col):
 for i from 0 to row - 1:
 // Check if another queen exists in the same column or diagonals
 if board[i] == col or
 board[i] - i == col - row or
 board[i] + i == col + row:
 return false
 return true

function print_solution(board):
 for i from 0 to 7:
 for j from 0 to 7:
 if board[i] == j:
 print("Q", end=" ") // Print queen symbol
 else:
```

```
 print(".", end=" ") // Print empty space
 print()

procedure eight_queens():
 board[8] = {-1, -1, -1, -1, -1, -1, -1, -1} // Initialize the chessboard

 if solve_8_queens(board, 0):
 print_solution(board) // Print the solution
 else:
 print("No solution exists")
```

Conclusion:

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____