**Assignment 6**

**Title:**

Implementation of Basic Search Strategies for Solving the 8-Queens Problem

**Aim:**

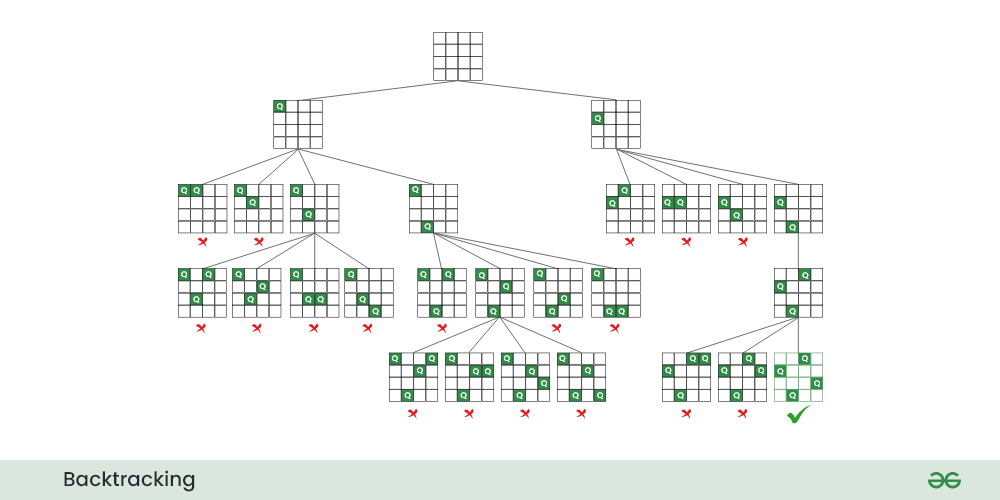
To implement basic search strategies like **Backtracking** and **Hill Climbing** for solving the **8-Queens Problem** and evaluate their effectiveness.

**Objectives:**

1. To understand the **8-Queens Problem** and the constraints involved.
2. To implement basic search strategies like **Backtracking** and **Hill Climbing** to find solutions to the 8-Queens Problem.
3. To explore the state-space representation and the methods to handle constraints while placing queens on the board.
4. To compare different search strategies based on their effectiveness and efficiency.

**Theory:**

**8-Queens Problem Overview:**



The **8-Queens Problem** is a classic constraint satisfaction problem where the goal is to place **8 queens** on an **8x8 chessboard** such that no two queens threaten each other. This means:

* No two queens can be placed in the same row, column, or diagonal.

The problem has multiple solutions and can be solved using various search strategies, such as **Backtracking**, **Hill Climbing**, **Genetic Algorithms**, etc.

**Search Strategies:**

1. **Backtracking:**
   * A **depth-first search (DFS)** approach that incrementally builds the solution.
   * At each level, it tries to place a queen in a safe position in the current row. If no valid position is found, it **backtracks** to the previous row and tries a different placement.
   * Ensures all constraints are met before moving to the next step.
2. **Hill Climbing:**
   * A **local search** algorithm where a single solution is modified to minimize the number of conflicts (queens attacking each other).
   * The algorithm makes small changes (moving queens) and evaluates the new state.
   * It can get stuck in **local minima**, so variants like **Simulated Annealing** or **Random Restart** can be used.

**State Space Representation:**

* The problem's state can be represented as a **list of integers**, where each integer represents the column position of the queen in that particular row. For example, the state [0, 4, 7, 5, 2, 6, 1, 3] means:
  + Queen 1 is placed at row 1, column 0.
  + Queen 2 is placed at row 2, column 4.
  + And so on.

**Procedure:**

**1. Define the Problem Space:**

* We have an 8x8 chessboard, and the goal is to place 8 queens such that no queen threatens another.
* Represent each state as a list of integers, where the index represents the row, and the value at each index represents the column position of the queen in that row.

**2. Implement Backtracking Search:**

* **Recursive Function**:
  + The algorithm places queens row by row.
  + For each row, it tries every column position.
  + Before placing a queen in a row-column position, it checks if the position is **safe** (no other queen in the same column, row, or diagonal).
  + If a safe position is found, the algorithm moves to the next row.
  + If no valid position is found in a row, it backtracks to the previous row and tries a different column.
* **Backtracking Algorithm Steps**:
  + Start with the first row.
  + For each column, check if placing a queen is safe.
  + If safe, move to the next row.
  + If no safe columns are available in a row, backtrack to the previous row and try a different position.
  + Repeat until all queens are placed or all possibilities are exhausted.

**3. Implement Hill Climbing Search:**

* **Initial State**:
  + Start with a random configuration of 8 queens (one in each row).
* **Move Generation**:
  + Generate neighboring states by moving a queen within its row.
* **Evaluation Function**:
  + For each state, calculate the number of conflicts between queens (queens attacking each other).
* **Hill Climbing Steps**:

1. Start with a random configuration of queens.
2. Evaluate the number of conflicts.
3. Move a queen to a new position within its row to reduce the number of conflicts.
4. If no better neighbor is found, the algorithm stops (may be a local minimum).
5. Variants like **Random Restart** can be used to overcome local minima by restarting the process with a new random state.

**4. Compare Search Strategies:**

* Implement both **Backtracking** and **Hill Climbing** strategies and compare their:
  + **Time complexity**.
  + **Efficiency** (number of states explored).
  + **Effectiveness** (whether a valid solution is found).
* Backtracking is guaranteed to find a solution (if one exists), while Hill Climbing may get stuck in local minima.

**Output:**

* The program outputs the final configuration of queens on the board where no two queens threaten each other.

**Expected Output:**

1. **Backtracking Solution**:
   * A valid configuration where 8 queens are placed on the board without conflicts. Example:

\_ Q \_ \_ \_ \_ \_ \_

\_ \_ \_ \_ Q \_ \_ \_

\_ \_ \_ \_ \_ \_ Q \_

Q \_ \_ \_ \_ \_ \_ \_

\_ \_ Q \_ \_ \_ \_ \_

\_ \_ \_ \_ \_ Q \_ \_

\_ \_ \_ Q \_ \_ \_ \_

\_ \_ \_ \_ \_ \_ \_ Q

**2. Hill Climbing Solution**:

* + A final configuration of queens after local optimization. If successful, it will also be a conflict-free solution.
  + If the algorithm gets stuck, the program will restart from a new random state.

**Procedure Example (Backtracking):**

1. **Start with an empty chessboard**.
   * Row 0: Try placing a queen in each column (0-7) and check if it's safe.
   * Row 1: Once a queen is placed in Row 0, move to Row 1 and try placing a queen in a safe position.
   * Continue this for all rows until a solution is found or backtrack if no valid position is found in a row.
2. **Backtrack when necessary**:
   * If no valid column is found for a row, backtrack to the previous row and try a different column for the queen placed there.
3. **Final Configuration**:
   * The result will be a configuration where no two queens threaten each other.

**Conclusion:**

In this lab, we successfully implemented two search strategies—**Backtracking** and **Hill Climbing**—to solve the **8-Queens Problem**. Backtracking is an exhaustive search strategy that guarantees finding a solution if one exists, while Hill Climbing is a local search strategy that works efficiently but can get stuck in local minima. By comparing the two approaches, we can observe their strengths and weaknesses in solving constraint satisfaction problems like the 8-Queens Problem.