**Assignment 6 :**

**Implementation of Basic Search Strategies: 8-Queens Problem**

**Aim:** To explore and implement basic search strategies for solving the 8-Queens problem, thereby enhancing understanding of backtracking and search algorithms in artificial intelligence.

**Objectives:**

1. To explore and implement basic search strategies for solving the 8-Queens problem.
2. To enhance understanding of backtracking and search algorithms in artificial intelligence.
3. To demonstrate the effectiveness of recursive algorithms in solving constraint satisfaction problems.

**Theory:**

The 8-Queens problem is a classic problem in computer science and artificial intelligence that involves placing eight queens on a standard chessboard such that no two queens threaten each other. This means that no two queens can share the same row, column, or diagonal. The problem is a classic example of constraint satisfaction, where solutions must adhere to specific rules.

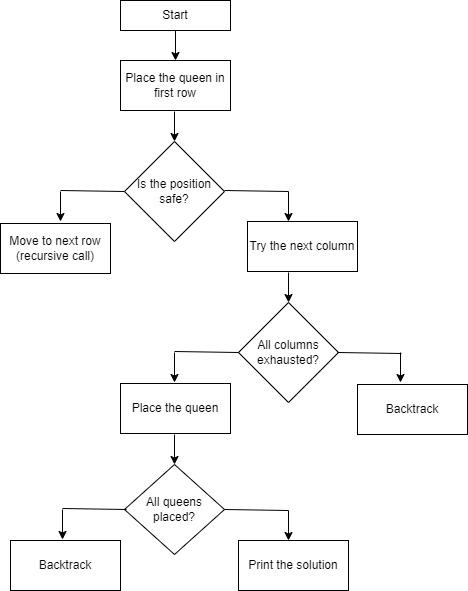
**Methodology:**

1. **Problem Representation**: Represent the chessboard as an 8x8 grid. Use an array to keep track of the positions of the queens, where the index represents the row and the value at that index represents the column.
2. **Backtracking Approach**:
   * Start from the first row and attempt to place a queen in each column of that row.
   * For each placement, check for conflicts with previously placed queens.
   * If a conflict arises, backtrack and try the next column in the previous row.
   * Continue this process recursively until all queens are placed or all possibilities are exhausted.
3. **Solution Collection**: Store all valid configurations where the queens are placed without threatening each other.

**Working Principle / Algorithm:**

1. **Initialize**: Create an empty chessboard and a function to check for valid queen placements.
2. **Recursive Function**:
   * For each row from 0 to 7:
     + For each column from 0 to 7:
       - Place a queen at (row, column).
       - Check if this placement is safe (i.e., no two queens threaten each other).
       - If safe, recursively call the function for the next row.
       - If placing leads to a solution, save the configuration.
       - If it leads to a conflict, remove the queen and try the next column.
3. **Base Case**: When all queens are placed successfully, return the solution.

**Workflow diagram:**

****

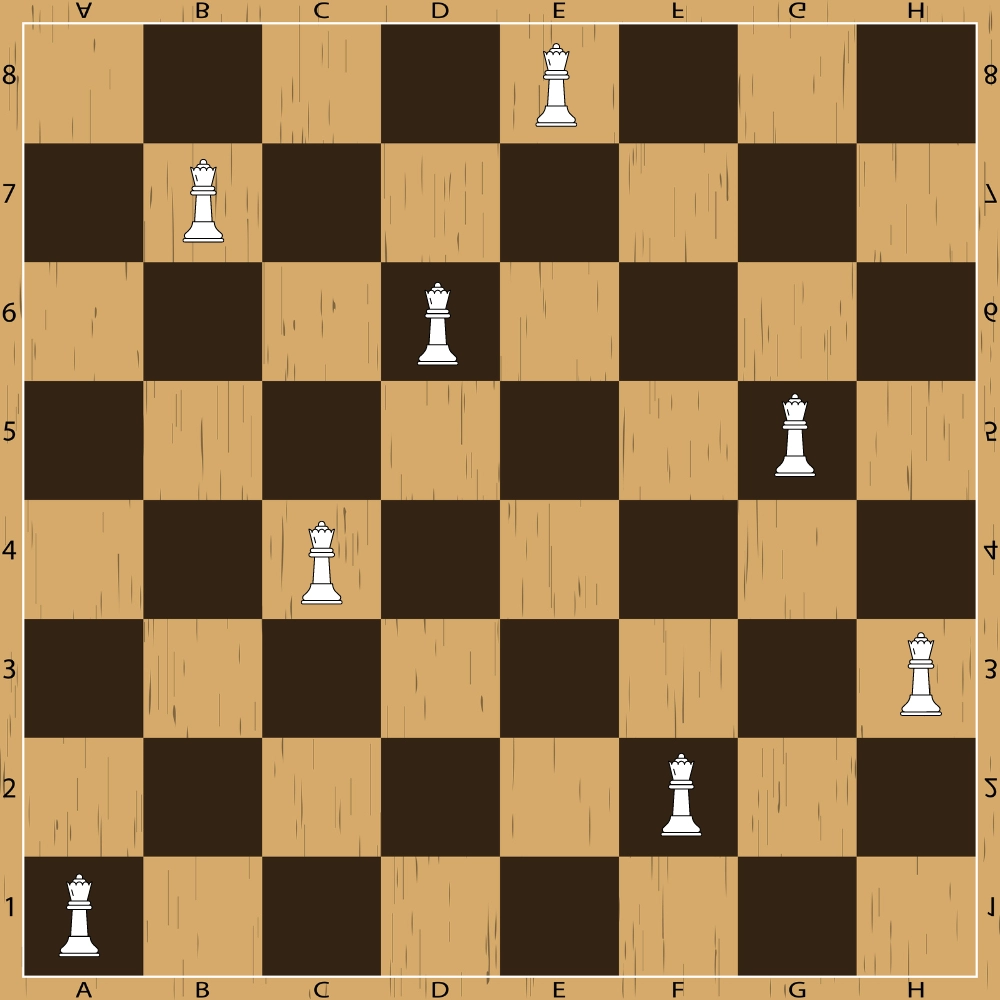
1. **Time complexity of Backtracking algorithm :**

The time complexity of the backtracking algorithm for the 8-Queens problem is O(N!), where N is the number of queens. This is due to the fact that there are N possible columns for the first queen, (N-1) for the second, and so on, resulting in a factorial growth in the number of configurations to explore. However, the backtracking approach prunes many of these configurations, making it more efficient than a naive brute-force search.

1. **Space complexity of Backtracking algorithm**

The space complexity of the backtracking algorithm is O(N), where N is the number of queens. This accounts for the recursion stack and the storage of the current state of the board (i.e., the placement of queens). The depth of the recursion corresponds to the number of queens being placed.

1. **Applications of Backtracking algorithm**
2. **Artificial Intelligence:** Used in solving puzzles and games that require constraint satisfaction.
3. **Graph Theory:** Applied in problems such as graph coloring and Hamiltonian paths.
4. **Scheduling:** Helps in scheduling tasks where constraints are involved.
5. **Combinatorial Optimization:** Used in problems like the traveling salesman problem and resource allocation.
6. **Advantages of Backtracking algorithm**
7. **Efficient Search:** It prunes the search space, reducing the number of configurations that need to be explored compared to exhaustive search methods.
8. **Flexibility:** The algorithm can be adapted to various problems with similar constraint structures.
9. **Simplicity:** The concept is straightforward and the implementation is relatively simple.
10. **Limitations of Backtracking algorithm**
11. **Time Complexity:** Although it is more efficient than brute force, backtracking can still exhibit exponential time complexity in the worst-case scenarios, particularly as the size of N increases.
12. **Space Complexity:** The algorithm requires additional memory for maintaining the recursion stack and the state of the board, which can be substantial for larger problems.
13. **No Guarantee of Optimal Solutions:** The algorithm finds valid solutions but does not ensure that these solutions are the best or optimal.
14. **Sample Example**



One possible solution to the 8 queens problem using backtracking is shown below. In the first row, the queen is at E8 square, so we have to make sure no queen is in column E and row 8 and also along its diagonals. Similarly, for the second row, the queen is on the B7 square, thus, we have to secure its horizontal, vertical, and diagonal squares. The same pattern is followed for the rest of the queens.

**Output -**

**0 0 0 0 1 0 0 0**

**0 1 0 0 0 0 0 0**

**0 0 0 1 0 0 0 0**

**0 0 0 0 0 0 1 0**

**0 0 1 0 0 0 0 0**

**0 0 0 0 0 0 0 1**

**0 0 0 0 0 1 0 0**

**1 0 0 0 0 0 0 0**

**Brute Force Approach -**

One brute-force approach to solving this problem is as follows:

* Generate all possible permutations of the numbers 1 to 8, representing the columns on the chessboard.
* For each permutation, check if it represents a valid solution by checking that no two queens are in the same row or diagonal.
* If a valid solution is found, print the board layout.

While this approach works for small numbers, it quickly becomes inefficient for larger sizes as the number of permutations to check grows exponentially. More efficient algorithms, such as backtracking or genetic algorithms, can be used to solve the problem in a more optimized way.

**Backtracking Approach**

This approach rejects all further moves if the solution is declined at any step, goes back to the previous step and explores other options.

### **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.

**Conclusion -**

The 8-Queens problem exemplifies the utility of basic search strategies in computer science, particularly through the application of backtracking. This approach effectively reduces the search space and provides solutions to complex constraint satisfaction problems. Understanding the principles behind these algorithms lays the groundwork for more advanced studies in artificial intelligence and algorithm design.