Aim: Write a program to implement 8 Queens Problem.

Theory:

Problem Statement
Given an 8x8 chess board, you must place 8 queens on the board so that no two queens attack each other. Print all possible matrices satisfying the conditions with positions with queens marked with '1' and empty spaces with '0'. You must solve the 8 queens problem using backtracking.
Note 1: A queen can move vertically, horizontally and diagonally in any number of steps.
Note 2: You can also go through the N-Queen Problem for the general approach to solving this problem.

What is Backtracking?
Backtracking the solution of the problem depends on the previous steps taken. We take a step and then analyze it that whether it will give the correct answer or not? And if not, then we move back and change the previous step.
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.

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.

Theory of the 8-Queen Problem

The **8-Queen Problem** is a classic combinatorial problem in computer science and artificial intelligence. It involves placing 8 queens on a standard chessboard (8×8 grid) in such a way that no two queens threaten each other. This means:

1. No two queens can be in the same row.
2. No two queens can be in the same column.
3. No two queens can be on the same diagonal.

Significance:

- **Constraint Satisfaction**: The problem is an example of a constraint satisfaction problem (CSP) and is often used to teach backtracking and recursion.
- **Search Optimization**: Techniques like branch-and-bound or heuristic search are applied to efficiently solve the problem.

Algorithm for Solving the 8-Queen Problem

Step-by-Step Approach:

1. **Define Constraints**:
    - Use a chessboard represented as a 2D array or a single array of size 8 (where the index represents the row and the value represents the column).
2. **Check for Valid Placement**:
    - Ensure that the current placement of a queen does not conflict with previously placed queens.
3. **Recursive Backtracking**:
    - Place a queen on a row, and recursively attempt to place queens on subsequent rows.
    - If a valid position is not found, backtrack to the previous row and try the next possible position.
4. **Base Case**:
    - If all 8 queens are placed successfully, print or store the solution.
5. **Optimization** (Optional):

- Use heuristics to speed up the search process by selecting rows or columns with fewer constraints.

Pseudocode for the 8-Queen Problem:

```
function solveNQueens(board, row):
    if row == 8:
        printSolution(board)
        return true

    for col in range(0, 8):
        if isSafe(board, row, col):
            board[row][col] = 1  # Place the queen
            if solveNQueens(board, row + 1):
                return true
            board[row][col] = 0  # Backtrack if needed
    return false

function isSafe(board, row, col):
    for i from 0 to row-1:
        if board[i][col] == 1:
            return false
    for i, j = row-1, col-1 while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return false
    for i, j = row-1, col+1 while i >= 0 and j < 8:
        if board[i][j] == 1:
            return false
    return true
```

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
```

Implementation of the Approach
Implementation in C++
To solve the 8 queens problem using backtracking

```
//Coding Ninjas
//C++ solution to 8 queens problem using backtracking
#include <bits/stdc++.h>
using namespace std;
int countt=0;

// A function to print a solution
void print(int board[][8]){
for(int i=0;i<8;i++){
for(int j=0;j<8;j++){
cout<<board[i][j]<<" ";
}
cout<<endl;
}
cout<<"-----------------\n";
}

//Function to check whether a position is valid or not
bool isValid(int board[][8],int row,int col){
//loop to check horizontal positions
for(int i=col;i>=0;i--){
if(board[row][i])
return false;
}
int i=row,j=col;
//loop to check the upper left diagonal
while(i>=0&&j>=0){
if(board[i][j])
return false;
i--;
j--;
}
i=row;
j=col;
//loop to check the lower left diagonal
while(i<8&&j>=0){
if(board[i][j])
```

```
return false;
i++;
j--;
}
return true;
}

//function to check all the possible solutions
void ninjaQueens(int board[][8],int currentColumn){
if(currentColumn>=8)
return;
//loop to cover all the columns
for(int i=0;i<8;i++){
if(isValid(board,i,currentColumn)){
board[i][currentColumn]=1;
if(currentColumn==7){
print(board);
countt++;
}
//recursively calling the function
ninjaQueens(board,currentColumn+1);
//backtracking
board[i][currentColumn]=0;
}
}
}
int main() {
//initial board situation
int board[8][8]={{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}};
ninjaQueens(board,0);
/* In total, 92 solutions exist for 8x8 board. This statement will verify our code*/
cout<<countt<<endl;
return 0;
}
```