Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

NAME: ALISTAIR SALDANHA
SAPID: 60009200024
BATCH: K1

Experiment 9

(Backtracking)

Aim: Implementation of 8 queen problem.

Theory:

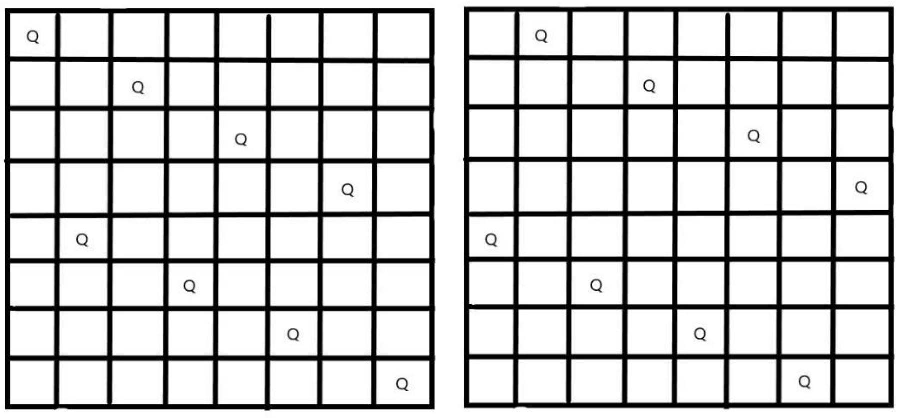
You are given an 8x8 chessboard, find a way to place 8 queens such that no queen can attack any other queen on the chessboard. A queen can only be attacked if it lies on the same row, or same column, or the same diagonal of any other queen. Print all the possible configurations.

To solve this problem, we will make use of the **Backtracking algorithm**. The backtracking algorithm, in general checks all possible configurations and test whether the required result is obtained or not. For thr given problem, we will explore all possible positions the queens can be relatively placed at. The solution will be correct when the number of placed queens = 8.

Input Format - the number 8, which does not need to be read, but we will take an input number for the sake of generalization of the algorithm to an NxN chessboard.

Output Format - all matrices that constitute the possible solutions will contain the numbers 0(for empty cell) and 1(for a cell where queen is placed). Hence, the output is a set of binary matrices.

Example:



Algorithm:

1. Begin from the leftmost column
2. If all the queens are placed, return true/ print the matrix
3. Check for all rows in the current column
    a) if queen placed safely, mark row and column; and recursively check if we approach in the current configuration, do we obtain a solution or not

   b) if placing yields a solution, return true
   c) if placing does not yield a solution, unmark and try other rows
4. If all rows tried and solution not obtained, return false and backtrack

Pseudocode:

```
Algorithm N_QUEEN (k, n)
// Description : To find the solution of n x n queen problem using backtracking
// Input :
n: Number of queen
k: Number of the queen being processed currently, initially set to 1.

// Output : n x 1 Solution tuple

for i ← 1 to n do
  if PLACE(k , i) then
    x[k] ← i
    if k == n then
      print X[1…n]
    else
      N_QUEEN(k + 1, n)
    end
  end
end

Function PLACE(k, i)
// k is the number of queen being processed
// i is the number of columns

for j ← 1 to k – 1 do
  if x[j] == i OR ((abs(x[j]) - i) == abs(j - k)) then
    return false
  end
end
return true
```

Time Complexity: O(N!)

Lab Assignment to Complete:

Using the above code and find a possible solution for N-queen problem, where N=4, 6, 8

Department of Computer Science and Engineering (Data Science)

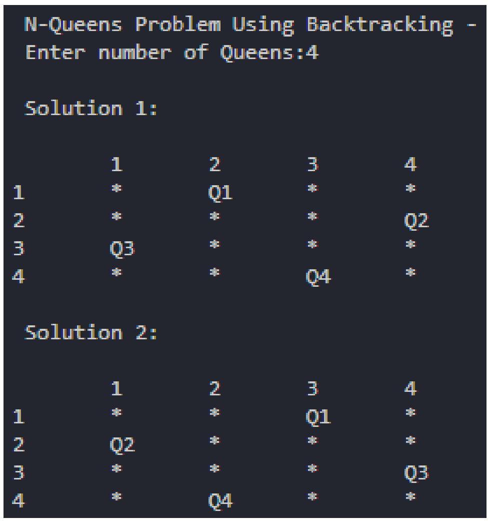
Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int board[MAX], count = 1;
// Display solution table
void print(int n)
{
    int i, j;
    printf("\n Solution %d:\n\n", count++);

    for (i = 1; i <= n; i++)
        printf("\t%d", i);
    for (i = 1; i <= n; i++)
    {
        printf("\n%d", i);
        for (j = 1; j <= n; j++)
        {
            if (board[i] == j)
                printf("\tQ%d", i);
            else
                printf("\t*");
        }
    }
    printf("\n");
}

int place(int row, int col)
{
    int i;
    // Checking the columns and diagonals of current Queen if no problem
return 0 else 1
    for (i = 1; i <= row - 1; i++)
    {
        //  Column condition        Diagonal Condition
        if ((board[i] == col) || (abs(board[i] - col) == abs(i - row)))
            return 0;
    }
    return 1;
}

// Positioning the Queen
void queen(int row, int n)
```

```
{
    int col;
    // row is nothing but queen as all queens should be placed in different
rows
    for (col = 1; col <= n; col++)
    {
        if (place(row, col))
        {
            board[row] = col; // no conflicts so place queen
            if (row == n)     // when all queens are placed
                print(n);     // print the board
            else              // else try queen for next position
                queen(row + 1, n);
        }
    }
}

int main()
{
    int n, i, j;
    printf(" N-Queens Problem Using Backtracking -");
    printf("\n Enter number of Queens:");
    scanf("%d", &n);
    queen(1, n);
    return 0;
}
```

Output:

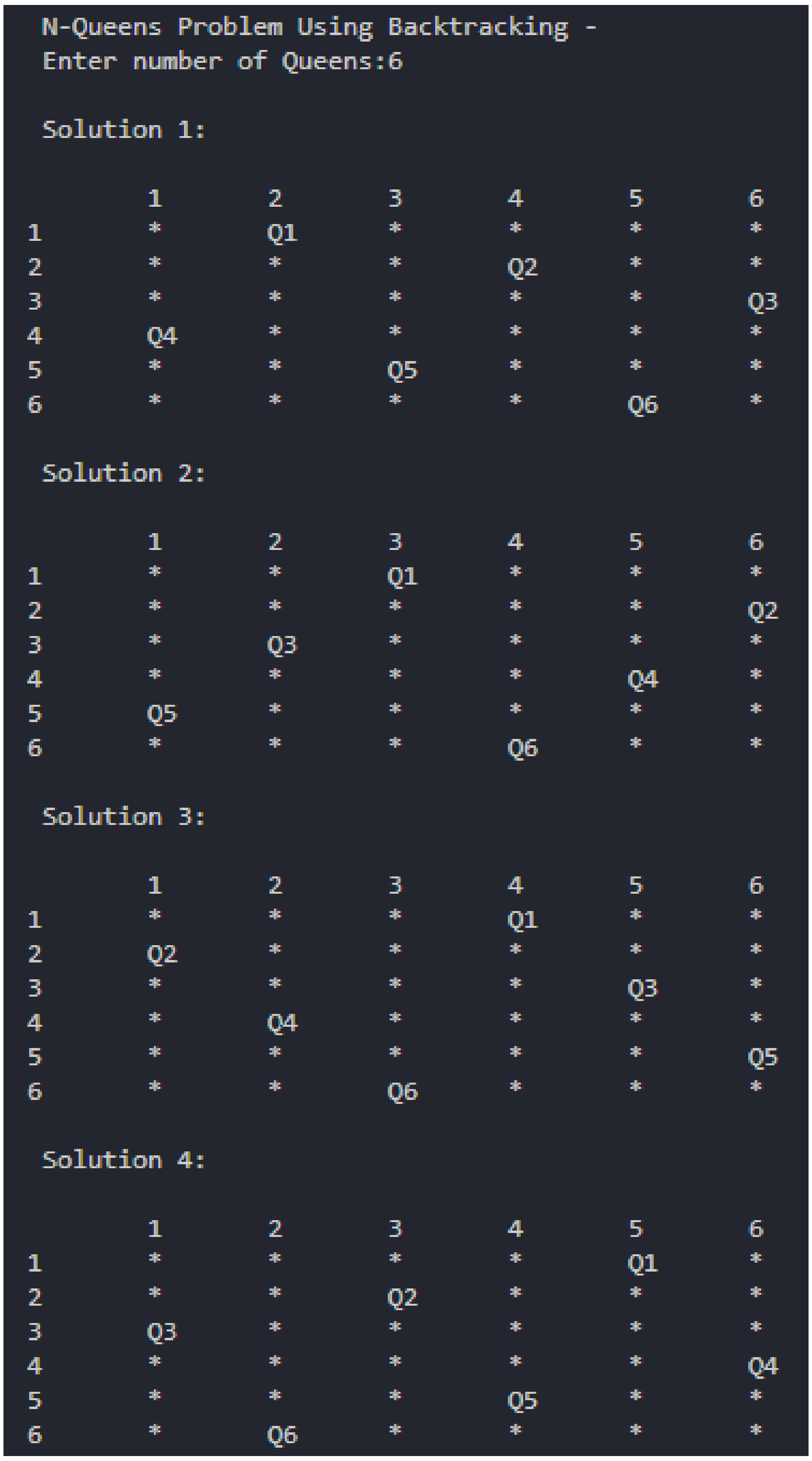
For 4-Queens



```
 N-Queens Problem Using Backtracking -
 Enter number of Queens:4

 Solution 1:

        1       2       3       4
1       *       Q1      *       *
2       *       *       *       Q2
3       Q3      *       *       *
4       *       *       Q4      *

 Solution 2:

        1       2       3       4
1       *       *       Q1      *
2       Q2      *       *       *
3       *       *       *       Q3
4       *       Q4      *       *
```

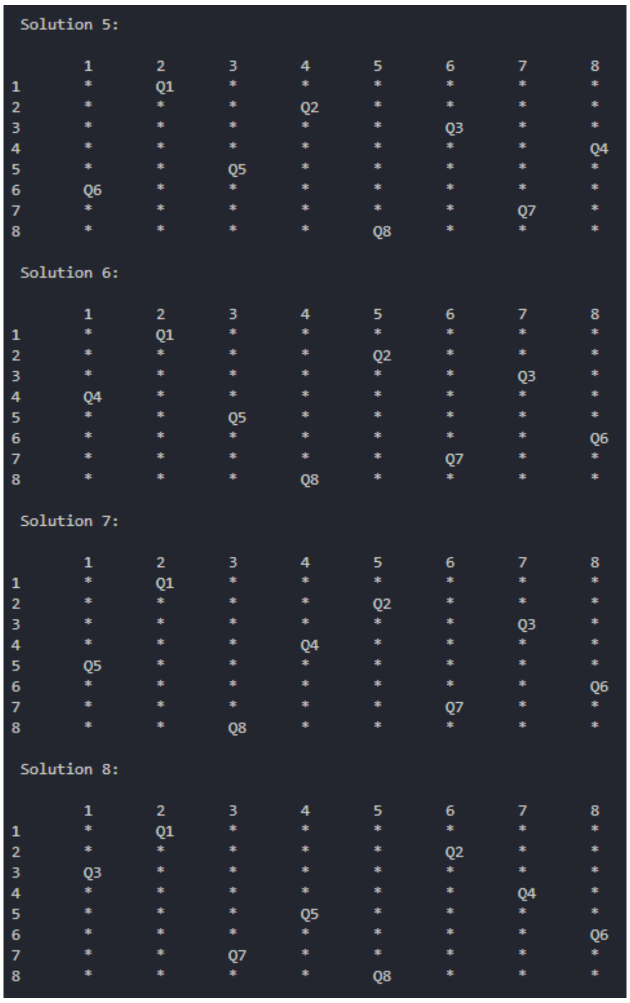
Department of Computer Science and Engineering (Data Science)

For 6-Queens



```
  N-Queens Problem Using Backtracking -
  Enter number of Queens:6

  Solution 1:

        1       2       3       4       5       6
1       *       Q1      *       *       *       *
2       *       *       *       Q2      *       *
3       *       *       *       *       *       Q3
4       Q4      *       *       *       *       *
5       *       *       Q5      *       *       *
6       *       *       *       *       Q6      *

  Solution 2:

        1       2       3       4       5       6
1       *       *       Q1      *       *       *
2       *       *       *       *       *       Q2
3       *       Q3      *       *       *       *
4       *       *       *       *       Q4      *
5       Q5      *       *       *       *       *
6       *       *       *       Q6      *       *

  Solution 3:

        1       2       3       4       5       6
1       *       *       *       Q1      *       *
2       Q2      *       *       *       *       *
3       *       *       *       *       Q3      *
4       *       Q4      *       *       *       *
5       *       *       *       *       *       Q5
6       *       *       Q6      *       *       *

  Solution 4:

        1       2       3       4       5       6
1       *       *       *       *       Q1      *
2       *       *       Q2      *       *       *
3       Q3      *       *       *       *       *
4       *       *       *       *       *       Q4
5       *       *       *       Q5      *       *
6       *       Q6      *       *       *       *
```

For 8-Queens



```
Solution 5:

        1       2       3       4       5       6       7       8
1       *       Q1      *       *       *       *       *       *
2       *       *       *       Q2      *       *       *       *
3       *       *       *       *       *       Q3      *       *
4       *       *       *       *       *       *       *       Q4
5       *       *       Q5      *       *       *       *       *
6       Q6      *       *       *       *       *       *       *
7       *       *       *       *       *       *       Q7      *
8       *       *       *       *       Q8      *       *       *

Solution 6:

        1       2       3       4       5       6       7       8
1       *       Q1      *       *       *       *       *       *
2       *       *       *       *       Q2      *       *       *
3       *       *       *       *       *       *       Q3      *
4       Q4      *       *       *       *       *       *       *
5       *       *       Q5      *       *       *       *       *
6       *       *       *       *       *       *       *       Q6
7       *       *       *       *       *       Q7      *       *
8       *       *       *       Q8      *       *       *       *

Solution 7:

        1       2       3       4       5       6       7       8
1       *       Q1      *       *       *       *       *       *
2       *       *       *       *       Q2      *       *       *
3       *       *       *       *       *       *       Q3      *
4       *       *       *       Q4      *       *       *       *
5       Q5      *       *       *       *       *       *       *
6       *       *       *       *       *       *       *       Q6
7       *       *       *       *       *       Q7      *       *
8       *       *       Q8      *       *       *       *       *

Solution 8:

        1       2       3       4       5       6       7       8
1       *       Q1      *       *       *       *       *       *
2       *       *       *       *       *       Q2      *       *
3       Q3      *       *       *       *       *       *       *
4       *       *       *       *       *       *       Q4      *
5       *       *       *       Q5      *       *       *       *
6       *       *       *       *       *       *       *       Q6
7       *       *       Q7      *       *       *       *       *
8       *       *       *       *       Q8      *       *       *
```