C++ Code :-

| #include<stdio.h> #include<string.h> #define N 8  /\* A utility function to print solution \*/ **void** **printSolution**(**int** board[N][N]) {  **for** (**int** i = 0; i < N; i++)  {  **for** (**int** j = 0; j < N; j++)  printf("%2d ", board[i][j]);  printf("\n");  } }  /\* A Optimized function to check if a queen can be placed on board[row][col] \*/ **int** **isSafe**(**int** row, **int** col, **int** slashCode[N][N],  **int** backslashCode[N][N],**int** rowLookup[],  **int** slashCodeLookup[],**int** backslashCodeLookup[] ) {  **if** (slashCodeLookup[slashCode[row][col]] ||  backslashCodeLookup[backslashCode[row][col]] ||  rowLookup[row])  **return** 0;   **return** 1; }  /\* A recursive utility function to solve N Queen problem \*/ **int** **solveNQueensUtil**(**int** board[N][N], **int** col,  **int** slashCode[N][N], **int** backslashCode[N][N],  **int** rowLookup[N],  **int** slashCodeLookup[],  **int** backslashCodeLookup[] ) {  /\* base case: If all queens are placed  then return true \*/  **if** (col >= N)  **return** 1;   /\* Consider this column and try placing  this queen in all rows one by one \*/  **for** (**int** i = 0; i < N; i++)  {  /\* Check if queen can be placed on  board[i][col] \*/  **if** ( isSafe(i, col, slashCode,  backslashCode, rowLookup,  slashCodeLookup, backslashCodeLookup) )  {  /\* Place this queen in board[i][col] \*/  board[i][col] = 1;  rowLookup[i] = 1;  slashCodeLookup[slashCode[i][col]] = 1;  backslashCodeLookup[backslashCode[i][col]] = 1;   /\* recur to place rest of the queens \*/  **if** ( solveNQueensUtil(board, col + 1,  slashCode, backslashCode,  rowLookup, slashCodeLookup, backslashCodeLookup) )  **return** 1;   /\* If placing queen in board[i][col]  doesn't lead to a solution, then backtrack \*/   /\* Remove queen from board[i][col] \*/  board[i][col] = 0;  rowLookup[i] = 0;  slashCodeLookup[slashCode[i][col]] = 0;  backslashCodeLookup[backslashCode[i][col]] = 0;  }  }   /\* If queen can not be place in any row in  this column col then return false \*/  **return** 0; }  /\* This function solves the N Queen problem using Branch and Bound. It mainly uses solveNQueensUtil() to solve the problem. It returns false if queens cannot be placed, otherwise return true and prints placement of queens in the form of 1s. Please note that there may be more than one solutions, this function prints one of the feasible solutions.\*/ **int** **solveNQueens**() {  **int** board[N][N];  memset(board, 0, **sizeof** board);   // helper matrices  **int** slashCode[N][N];  **int** backslashCode[N][N];   // arrays to tell us which rows are occupied  **int** rowLookup[N] = {0};   //keep two arrays to tell us  // which diagonals are occupied  **int** slashCodeLookup[2\*N - 1] = {0};  **int** backslashCodeLookup[2\*N - 1] = {0};   // initialize helper matrices  **for** (**int** r = 0; r < N; r++)  **for** (**int** c = 0; c < N; c++) {  slashCode[r][c] = r + c,  backslashCode[r][c] = r - c + 7;  }   **if** (solveNQueensUtil(board, 0,  slashCode, backslashCode,  rowLookup, slashCodeLookup, backslashCodeLookup) ==  0 )  {  printf("Solution does not exist");  **return** 0;  }    // solution found  printSolution(board);  **return** 1; }   // Driver program to test above function **int** **main**() {  solveNQueens();    **return** 0; } |
| --- |

Output:-

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 1 0 0

Python Code:-

#

| Python program **to** solve N Queen # Problem using backtracking   global N N = 4   def printSolution(board):  **for** i **in** **range**(N):  **for** j **in** **range**(N):  print (board[i][j],**end**=' ')  print()     # A utility **function** **to** check **if** a queen can # be placed **on** board[row][col]. **Note** that this # **function** **is** called **when** "col" queens are # already placed **in** columns from 0 **to** col -1. # So we need **to** check only left **side** **for** # attacking queens def isSafe(board, row, col):    # Check this row **on** left **side**  **for** i **in** **range**(col):  **if** board[row][i] == 1:  **return** **False**    # Check upper diagonal **on** left **side**  **for** i, j **in** zip(**range**(row, -1, -1), **range**(col, -1, -1)):  **if** board[i][j] == 1:  **return** **False**    # Check lower diagonal **on** left **side**  **for** i, j **in** zip(**range**(row, N, 1), **range**(col, -1, -1)):  **if** board[i][j] == 1:  **return** **False**    **return** **True**   def solveNQUtil(board, col):  # base **case**: **If** **all** queens are placed  # **then** **return** **true**  **if** col >= N:  **return** **True**    # Consider this column **and** try placing  # this queen **in** **all** rows one by one  **for** i **in** **range**(N):    **if** isSafe(board, i, col):  # Place this queen **in** board[i][col]  board[i][col] = 1    # recur **to** place rest **of** the queens  **if** solveNQUtil(board, col + 1) == **True**:  **return** **True**    # **If** placing queen **in** board[i][col  # doesn't lead **to** a solution, **then**  # queen from board[i][col]  board[i][col] = 0    # **if** the queen can **not** be placed **in** any row **in**  # this column col **then** **return** **false**  **return** **False**   # This **function** solves the N Queen problem using # Backtracking. It mainly uses solveNQUtil() **to** # solve the problem. It returns **false** **if** queens # cannot be placed, otherwise **return** **true** **and** # placement **of** queens **in** the form **of** 1s. # **note** that there may be more than one # solutions, this **function** prints one **of** the # feasible solutions. def solveNQ():  board = [ [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ]    **if** solveNQUtil(board, 0) == **False**:  print ("Solution does not exist")  **return** **False**    printSolution(board)  **return** **True**   # driver program **to** test above **function** solveNQ() |
| --- |

Output:-

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0