12. Design and implement C Program for N Queen's problem using Backtracking.

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Function to print the solution

void printSolution(int \*\*board, int N)

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

printf("%s ", board[i][j] ? "Q" : "#");

}

printf("\n");

}

}

// Function to check if a queen can be placed on board[row][col]

bool isSafe(int \*\*board, int N, int row, int col)

{

int i, j;

// Check this row on left side

for (i = 0; i < col; i++)

{

if (board[row][i])

{

return false;

}

}

// Check upper diagonal on left side

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

{

if (board[i][j])

{

return false;

}

}

// Check lower diagonal on left side

for (i = row, j = col; j >= 0 && i < N; i++, j--)

{

if (board[i][j])

{

return false;

}

}

return true;

}

// A recursive utility function to solve N Queen problem

bool solveNQUtil(int \*\*board, int N, int col)

{

// 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 (int i = 0; i < N; i++)

{

if (isSafe(board, N, i, col))

{

// Place this queen in board[i][col]

board[i][col] = 1;

// Recur to place rest of the queens

if (solveNQUtil(board, N, col + 1))

{

return true;

}

// If placing queen in board[i][col] doesn't lead to a solution,

// then remove queen from board[i][col]

board[i][col] = 0; // BACKTRACK

}

}

// If the queen cannot 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 prints the placement of queens

bool solveNQ(int N)

{

int \*\*board = (int \*\*)malloc(N \* sizeof(int \*));

for (int i = 0; i < N; i++)

{

board[i] = (int \*)malloc(N \* sizeof(int));

for (int j = 0; j < N; j++)

{

board[i][j] = 0;

}

}

if (!solveNQUtil(board, N, 0))

{

printf("Solution does not exist\n");

for (int i = 0; i < N; i++)

{

free(board[i]);

}

free(board);

return false;

}

printSolution(board, N);

for (int i = 0; i < N; i++)

{

free(board[i]);

}

free(board);

return true;

}

int main()

{

int N;

printf("Enter the number of queens: ");

scanf("%d", &N);

solveNQ(N);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT-1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the number of queens: 4

# # Q #

Q # # #

# # # Q

# Q # #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT-2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the number of queens: 3

Solution does not exist