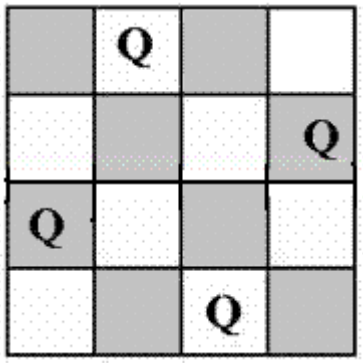
Printing all solutions in N-Queen Problem

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



In previous post, we have discussed an approach that prints only one possible solution, so now in this post the task is to print all solutions in N-Queen Problem. The solution discussed here is an extension of same approach.

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Backtracking Algorithm

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.  Do following
   for every tried row.
    a) If the queen can be placed safely in this row
       then mark this [row, column] as part of the
       solution and recursively check if placing
       queen here leads to a solution.
    b) If placing queen in [row, column] leads to a
       solution then return true.
    c) If placing queen doesn't lead to a solution
       then unmark this [row, column] (Backtrack)
       and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked,
   return false to trigger backtracking.
```

There is only a slight modification in above algorithm that is highlighted in the code.

```
/* C/C++ program to solve N Queen Problem using
backtracking */
#define N 4
#include<stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    static  int k = 1;
    printf("%d-\n",k++);
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}

/* 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 */
bool isSafe(int board[N][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[N][N], int col)
{
    /* base case: If all queens are placed
    then return true */
    if (col == N )
    {
        printSolution(board);
        return true;
    }

    /* 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(board, i, col) )
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            solveNQUtil(board, col + 1) ;

            // below commented statement is replaced
            // by above one
            /* if ( solveNQUtil(board, 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 queen can not be place 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 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.*/
void solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
    };

    if (solveNQUtil(board, 0))
    {
        printf("Solution does not exist");
        return ;
    }

    return ;
```

```
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

Run on IDE

Output:

```
1-
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0

2-
 0  1  0  0
 0  0  0  1
 1  0  0  0
 0  0  1  0
```

This article is contributed by **Sahil Chhabra (akku)**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Backtracking

Recommended Posts:

Backtracking | Set 3 (N Queen Problem)
Backtracking | Set 7 (Sudoku)
Backtracking | Set 4 (Subset Sum)
Backtracking | Set 2 (Rat in a Maze)
Check if a given string is sum-string
Combinational Sum

Combinations where every element appears twice and distance between appearances is equal to the value
Top 20 Backtracking Algorithm Interview Questions
Warnsdorff's algorithm for Knight's tour problem
Print all palindromic partitions of a string

(Login to Rate)

3.6 Average Difficulty : **3.6/5.0**
Based on **11** vote(s)

Basic | Easy | Medium | Hard | Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!

8 Comments **GeeksforGeeks** **Login**

Recommend **Share** **Sort by Newest**

Join the discussion…

LOG IN WITH **OR SIGN UP WITH DISQUS**

Name

Carlos Eduardo Briceño • a month ago

I walked through this code and found the search could be done 2x faster if you write separate code to deal with column 0. You don't need to inspect all squares on column 0 like you do just to find solutions that are just symmetries of solutions previously found. Please consider:

```
/* A utility function to print solution */
void printSolution(int board[N][N])
{
static int k = 1;
printf("%d-\n",k++);
for (int i = 0; i < N; i++)
{
for (int j = 0; j < N; j++)
printf(" %d ", board[i][j]);
printf("\n");
}
```

```
printf("\n");
printf("%d-\n",k++);
for (int i = 0; i < N; i++)
```

see more

• Reply • Share ›

kalavala harsha • 5 months ago

it doesnt print solution doesnt exist in any case.for example take 3 *3 and 2*2 i

• Reply • Share ›

anand kumar ➚ kalavala harsha • 3 months ago

bro N queen problem has no solution for n=2 and n=3(try it yourself on paper) and for n>=4 the code is working absolutely fine.

1 • Reply • Share ›

harry ➚ kalavala harsha • 4 months ago

This is just to explain 4*4 in detail of how backtracking is used. Its hard coded like that (#define N 4) . Why should it print what you ask.?

• Reply • Share ›

kalavala harsha ➚ harry • 4 months ago

make it # define 2 and check whether it prints solution doesnt exists or not.as 2*2 has no solution whatever may be the input.

• Reply • Share ›

rohit soneta ➚ kalavala harsha • 4 months ago

yes it is not printing .... as in any case function is not returning value other than false...

• Reply • Share ›

Somya Srivastava • 5 months ago

what is the time complexity for this algorithm ?

• Reply • Share ›

HokageDatteBayo ➚ Somya Srivastava • 3 months ago

exponential

• Reply • Share ›