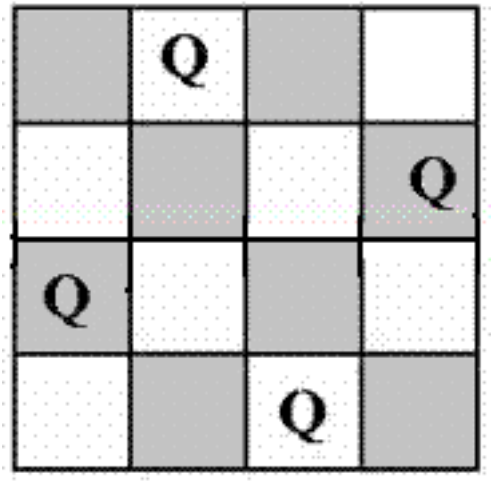
Backtracking | Set 3 (N Queen Problem)

We have discussed Knight's tour and Rat in a Maze problems in Set 1 and Set 2 respectively. Let us discuss N Queen as another example problem that can be solved using Backtracking.

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 expected output is a binary matrix which has 1s for the blocks where queens are placed. For example following is the output matrix for above 4 queen solution.

```
{ 0,  1,  0,  0}
{ 0,  0,  0,  1}
{ 1,  0,  0,  0}
{ 0,  0,  1,  0}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Naive Algorithm

Generate all possible configurations of queens on board and print a configuration

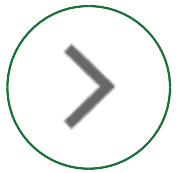
that satisfies the given constraints.

```
while there are untried conflagrations
{
   generate the next configuration
   if queens don't attack in this configuration then
   {
      print this configuration;
   }
}
```

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 tr
ied row.
    a) If the queen can be placed safely in this row then mark th
is [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 umark thi
s [row,
        column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false t
o trigger
    backtracking.
```

Implementation of Backtracking solution

C/C++ Java

```c
/* 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])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        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
```

Recent Comments

Popular Posts

- Top 10 Algorithms and Data Structures for Competitive Programming
- Top 10 algorithms in Interview Questions
- How to begin with Competitive Programming?
- Step by Step Guide for Placement Preparation
- Reflection in Java
- Memory Layout of C Programs
- Heavy Light Decomposition

```
   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)
        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] */
```

- Heavy Light Decomposition
- Sorted Linked List to Balanced BST
- Generics in Java
- Aho-Corasick Algorithm for Pattern Searching
- Insertion Sort , Binary Search , QuickSort , MergeSort , HeapSort

- Common Interview Puzzles
- Interview Experiences
- Advanced Data Structures
- Design Patterns
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Geometric Algorithms
- Searching
- Sorting
- Analysis of Algorithms
- Mathematical Algorithms
- Randomized Algorithms
- Recursion
- Game Theory

```
        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) )
                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 colum 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.*/
bool 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) ==  false )
    {
      printf("Solution does not exist" );
```

Tags

Adobe Advanced Data Structure Amazon Arrays Articles C-Output C/C++ Puzzles cpp-library Divide and Conquer Dynamic Programming Experienced Flipkart GATE GATE-CS-DS-&-Algo GBlog Graph Greedy Hash Internship Interview Experiences Java-Collections Linked Lists Mathematical Matrix MCQ Oracle Output Pattern Searching Project Python Searching sieve Sorting Stack STL Strings subsequence XOR

```
        return false;
    }

    printSolution(board);
    return true;
}

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

Run on IDE

Output: The 1 values indicate placements of queens

```
0  0  1  0
1  0  0  0
0  0  0  1
0  1  0  0
```

Sources:

http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf

http://en.literateprograms.org/Eight_queens_puzzle_%28C%29

http://en.wikipedia.org/wiki/Eight_queens_puzzle

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

GATE CS Corner Practice

Company Wise Coding

Backtracking Backtracking

Related Posts:

- Print all palindromic partitions of a string
- Word Break Problem using Backtracking
- Partition of a set into K subsets with equal sum

- Remove Invalid Parentheses
- Find shortest safe route in a path with landmines
- Longest Possible Route in a Matrix with Hurdles
- Match a pattern and String without using regular expressions
- Find Maximum number possible by doing at-most K swaps

<< Previous Post

Next Post >>

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **63** vote(s)

Add to TODO List

Mark as DONE

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

Load Comments

Share this post !