8 queen problem

The eight queens problem is the problem of placing eight queens on an 8×8 chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). More generally, the n queens problem places n queens on an n×n chessboard. There are different solutions for the problem. Backtracking | Set 3 (N Queen Problem) Branch and Bound | Set 5 (N Queen Problem) You can find detailed solutions at http://en.literateprograms.org/Eight_queens_puzzle_(C)

Explanation:

- This **pseudocode uses a backtracking algorithm** to find a solution to the 8 Queen problem, which consists of placing 8 queens on a chessboard in such a way that no two queens threaten each other.
- The algorithm starts by placing a queen on the first column, then it proceeds to the next column and places a queen in the **first safe** row of that column.
- If the algorithm reaches the 8th column and all **queens are placed in a safe position**, it prints the board and returns true.
  If the algorithm is unable to place a queen in a safe position in a certain column, it backtracks to the previous column and tries a different row.
- The “isSafe” function **checks** if it is safe to place a queen on a certain row and column by checking if there are any queens in the same row, diagonal or anti-diagonal.
- It’s worth to notice that this is just a high-level **pseudocode** and it might need to be adapted depending on the specific implementation and language you are using.

Here is an example of pseudocode for solving the 8 Queen problem using backtracking:

C++

```
#include <iostream>
#include <vector>
```

```
bool isSafe(vector<vector<int> >& board, int row, int col)
{
    for (int x = 0; x < col; x++)
        if (board[row][x] == 1)
            return false;
    for (int x = row, y = col; x >= 0 && y >= 0; x--, y--)
        if (board[x][y] == 1)
            return false;
    for (int x = row, y = col; x < N && y >= 0; x++, y--)
        if (board[x][y] == 1)
            return false;
    return true;
}

bool solveNQueens(vector<vector<int> >& board, int col)
{
    if (col == N) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                cout << board[i][j] << " ";
            cout << endl;
        }
        cout << endl;
        return true;
    }
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQueens(board, col + 1))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}

int main()
{
    vector<vector<int> > board(N, vector<int>(N, 0));
    if (!solveNQueens(board, 0))
        cout << "No solution found";
    return 0;
}
```

Java

```
import java.util.Arrays;
class GFG {

    static final int N = 8;

    // function to check if it is safe to place
```

```
    {

        // check if there is a queen in the same row to the
        // left
        for (int x = 0; x < col; x++)
            if (board[row][x] == 1)
                return false;

        // check if there is a queen in the upper left
        // diagonal
        for (int x = row, y = col; x >= 0 && y >= 0;
             x--, y--)
            if (board[x][y] == 1)
                return false;

        // check if there is a queen in the lower left
        // diagonal
        for (int x = row, y = col; x < N && y >= 0;
             x++, y--)
            if (board[x][y] == 1)
                return false;

        // if there is no queen in any of the above
        // positions, then it is safe to place a queen
        return true;
    }

    // function to solve the N-queens problem using
    // backtracking
    static boolean solveNQueens(int[][] board, int col)
    {

        // if all queens are placed, print the board
        if (col == N) {
            for (int[] row : board)
                System.out.println(Arrays.toString(row));
            System.out.println();
            return true;
        }

        // try placing a queen in each row of the current
        // column
        for (int i = 0; i < N; i++) {
            if (isSafe(board, i, col)) {
                board[i][col] = 1; // place the queen
                if (solveNQueens(board, col + 1))
                    return true;

                // backtrack if placing the queen doesn't
                // lead to a solution
                board[i][col] = 0;
            }
        }
```

```
    // initialize the board with zeros and call the
    // solveNQueens function to solve the problem
    public static void main(String[] args)
    {
        int[][] board = new int[N][N];
        if (!solveNQueens(board, 0))
            System.out.println("No solution found");
    }
}
```

Python3

```
N = 8 # (size of the chessboard)

def solveNQueens(board, col):
    if col == N:
        print(board)
        return True
    for i in range(N):
        if isSafe(board, i, col):
            board[i][col] = 1
            if solveNQueens(board, col + 1):
                return True
            board[i][col] = 0
    return False

def isSafe(board, row, col):
    for x in range(col):
        if board[row][x] == 1:
            return False
    for x, y in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[x][y] == 1:
            return False
    for x, y in zip(range(row, N, 1), range(col, -1, -1)):
        if board[x][y] == 1:
            return False
    return True

board = [[0 for x in range(N)] for y in range(N)]
if not solveNQueens(board, 0):
    print("No solution found")
```

C#

```
using System;
using System.Linq;

class GFG {

  static readonly int N = 8;
```

```
// a queen at a particular position
static bool isSafe(int[,] board, int row, int col)
{

  // check if there is a queen in the same row to the
  // left
  for (int x = 0; x < col; x++)
    if (board[row, x] == 1)
      return false;

  // check if there is a queen in the upper left
  // diagonal
  for (int x = row, y = col; x >= 0 && y >= 0;
       x--, y--)
    if (board[x, y] == 1)
      return false;

  // check if there is a queen in the lower left
  // diagonal
  for (int x = row, y = col; x < N && y >= 0;
       x++, y--)
    if (board[x, y] == 1)
      return false;

  // if there is no queen in any of the above
  // positions, then it is safe to place a queen
  return true;
}

// function to solve the N-queens problem using
// backtracking
static bool solveNQueens(int[,] board, int col)
{

  // if all queens are placed, print the board
  if (col == N) {
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
        Console.Write(board[i, j] + " ");
      }
      Console.WriteLine();
    }
    Console.WriteLine();
    return true;
  }

  // try placing a queen in each row of the current
  // column
  for (int i = 0; i < N; i++) {
    if (isSafe(board, i, col)) {
      board[i, col] = 1; // place the queen
      if (solveNQueens(board, col + 1))
        return true;
```

```
      }
    }

    // if no solution is found, return false
    return false;
  }

  // initialize the board with zeros and call the
  // solveNQueens function to solve the problem
  public static void Main(string[] args)
  {
    int[,] board = new int[N, N];
    if (!solveNQueens(board, 0))
      Console.WriteLine("No solution found");
  }
}
```

Javascript

```
const N = 8;

// function to check if it is safe to place a queen at a particular position
function isSafe(board, row, col) {
  // check if there is a queen in the same row to the left
  for (let x = 0; x < col; x++) {
    if (board[row][x] == 1) {
      return false;
    }
  }

  // check if there is a queen in the upper left diagonal
  for (let x = row, y = col; x >= 0 && y >= 0; x--, y--) {
    if (board[x][y] == 1) {
      return false;
    }
  }

  // check if there is a queen in the lower left diagonal
  for (let x = row, y = col; x < N && y >= 0; x++, y--) {
    if (board[x][y] == 1) {
      return false;
    }
  }

  // if there is no queen in any of the above positions, then it is safe to place a qu
  return true;
}

// function to solve the N-queens problem using backtracking
function solveNQueens(board, col) {
  // if all queens are placed, print the board
  if (col == N) {
    for (let i = 0; i < N; i++) {
```

```
    console.log("\n");
    return true;
  }

  // try placing a queen in each row of the current column
  for (let i = 0; i < N; i++) {
    if (isSafe(board, i, col)) {
      board[i][col] = 1; // place the queen
      if (solveNQueens(board, col + 1)) {
        return true;
      }
      board[i][col] = 0; // backtrack if placing the queen doesn't lead to a solution
    }
  }

  // if no solution is found, return false
  return false;
}

// initialize the board with zeros and call the solveNQueens function to solve the pro
let board = Array(N)
  .fill()
  .map(() => Array(N).fill(0));

if (!solveNQueens(board, 0)) {
  console.log("No solution found");
}
```

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], [0, 0, 1, 0, 0, 0, 0, 0]]
```

Time Complexity : O((m + g) log^2 n)

Last Updated : 08 Mar, 2023

31

Similar Reads

1. Find position of Queen in chessboard from given attack lines of queen
2. N-Queen Problem | Local Search using Hill climbing with random neighbour
3. Printing all solutions in N-Queen Problem
4. N Queen Problem using Branch And Bound
5. N Queen Problem | Backtracking-3
6. Check if a Queen can attack a given cell on chessboard
7. Count positions in a chessboard that can be visited by the Queen which are not visited by the King
8. Number of cells a queen can move with obstacles on the chessboard
9. N Queen in O(n) space
10. Nuts & Bolts Problem (Lock & Key problem) using Quick Sort

Related Tutorials

1. Learn Data Structures with Javascript | DSA Tutorial
2. Introduction to Max-Heap – Data Structure and Algorithm Tutorials
3. Introduction to Set – Data Structure and Algorithm Tutorials
4. Introduction to Map – Data Structure and Algorithm Tutorials
5. What is Dijkstra’s Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : lakshyadeeppatel, prasad264, unstoppablepandu, prajwalkandekar123, anikettchpiow, mallelagowtalm

Article Tags : chessboard-problems, Backtracking, DSA

Practice Tags : Backtracking

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org