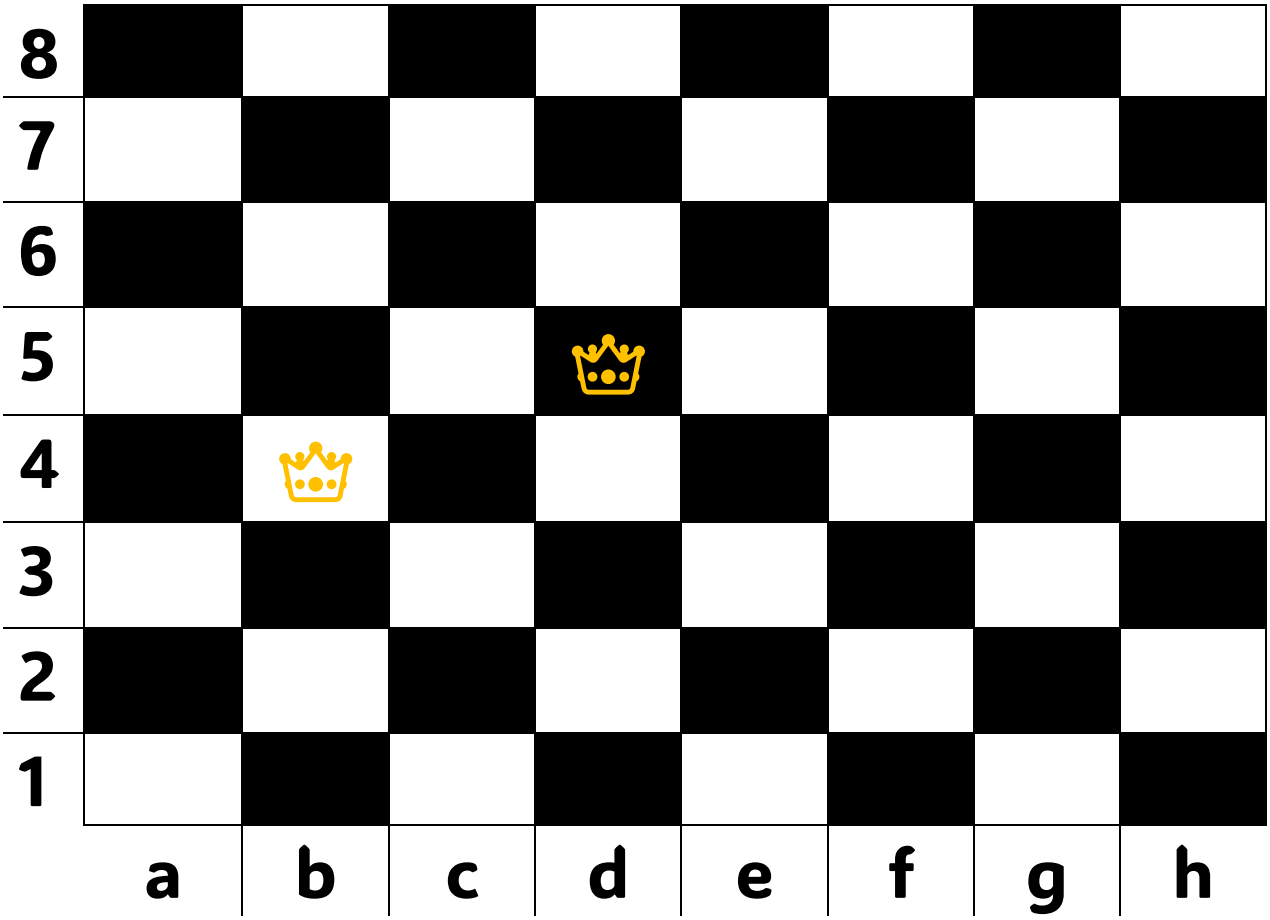
1) Description of the problem

What is N-Queens ?

An *n-queens configuration* is a set of *n* queens on an N*N chessboard such that no two are contained in the same row, column, or diagonal. The *n-queens problem* is to determine $Q(n)$, the number of distinct *n*-queens configurations. The 8-queens problem was first published by German chess composer Max Bezzel in 1848, and attracted the attention of many mathematicians, including Gauss.

In 1850, Nauck posed the problem with two queens on b4 and d5 already given. Is it possible to add six more queens to obtain an 8-queens configuration?



Given that two queens are already placed on b4 and d5, to achieve a valid 8-queens configuration, we need to place six more queens on the board such that no two queens threaten each other along rows, columns, or .diagonals

Let's analyze the situation :

The queens on b4 and d5 are positioned on the same diagonal.

A valid configuration requires no two queens to be in the same row, column, or diagonal.

As the two initial queens are on the same diagonal, they cover two diagonals entirely. To add six more queens without any conflicts, each new queen must occupy a different row, column, or diagonal than the existing queens.

However, given that the two initial queens cover two diagonals, it is impossible to place six more queens on the board without violating the constraints of the 8-queens problem. This configuration would result in a scenario where there are insufficient remaining spaces on the board for the required six queens without any conflicts.

Therefore, it's impossible to add six more queens to the existing configuration of two queens on b4 and d5 to obtain a valid 8-queens configuration.

2) Solution methods

The solution to the N-Queens problem involves placing N chess queens on an N×N chessboard in such a way that no two queens threaten each other. Queens can move horizontally, vertically, and diagonally without restrictions in chess, making the task challenging.

Several algorithms and techniques can solve the N-Queens problem :

Brute Force: Check all possible configurations by placing queens on the board and testing if they attack each other. This method can be slow and computationally expensive, especially for larger N.

Backtracking: Employ a recursive approach to explore potential configurations while backtracking when encountering conflicts. This method efficiently prunes the search space by abandoning branches that lead to conflicts.

Constraint Satisfaction: Use techniques like constraint propagation and constraint satisfaction to iteratively place queens while ensuring they do not violate any constraints (no two queens in the same row, column, or diagonal).

Optimization Techniques: Employ optimization algorithms like simulated annealing, genetic algorithms, or local search to efficiently find solutions by iteratively improving the placement of queens.

Bitmasking and Bitwise Operations: Implement efficient data structures and bitwise operations to represent the board state and optimize the checks for row, column, and diagonal conflicts.

Each approach has its advantages and is suited for different scenarios. Backtracking is a common and effective method for solving the N-Queens problem, but for larger N, more optimized algorithms might be preferred.

The goal of any solution method is to systematically explore the chessboard, considering various queen placements while ensuring none threaten each other. The challenge lies in efficiently searching through the vast solution space to find valid configurations without conflicts for a given N.

3) Algorithm

We used a Backtracking solution

What is backtracking ?

Backtracking is finding the solution of a problem whereby the solution depends on the previous steps taken. For example, in a maze problem, the solution depends on all the steps you take one-by-one.

If any of those steps is wrong, then it will not lead us to the solution.

In a maze problem, we first choose a path and continue moving along it. But once we understand that the particular path is incorrect, then we just come back and change it.

This is what backtracking basically is.

In backtracking, we first take a step and then we see if this step taken is correct or not i.e., whether it will give a correct answer or not. And if it doesn't, then we just come back and change our first step. In general, this is accomplished by recursion. Thus, in backtracking, we first start with a partial sub-solution of the problem (which may or may not lead us to the solution) and then check if we can proceed further with this sub-solution .or not. If not, then we just come back and change it

Thus, the general steps of backtracking are:

start with a sub-solution

check if this sub-solution will lead to the solution or not

If not, then come back and change the sub-solution and continue again

N queens on NxN chessboard

One of the most common examples of the backtracking is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen.

A queen can attack horizontally, vertically, or diagonally.

The solution to this problem is also attempted in a similar way.

We first place the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left.

If no safe place is left, then we change the position of the previously placed queen.

4) Analysis of algorithm

Code :

```java
private boolean isSafe(int[][] board , int column , int row){
        // check column from current cell to above
        for(int i=0 ; i<board.length ; i++){
            if(board[i][column]==1)
                return false;
        }
        // check upper right diagonal from current cell to above
        for(int i=column+1,j=row-1 ; i<board.length&&j>=0 ; i++,j--)
            if(board[j][i]==1)
                return false;
        // check upper left diagonal from current cell to above
        for(int i=column-1,j=row-1 ; i>=0&&j>=0 ; i--,j--)
            if(board[j][i]==1)
                return false;
        return true;
    }
    private boolean solve(int[][] board , int row){
        // check if we reach to end of board then we founded solution
        if(row==board.length)
            return true;
        // for loop all rows to set ecah safe cell to 1
        for(int i=0 ; i<board.length ; i++){
            // check if cell is safe
            if(isSafe(board,i,row)){
                // set cell to 1
                board[row][i]=1;
                // path col+1 to function
                if(solve(board,row+1))
                    return true;
                // if above "if false" then we backtrak and rest cell to
                board[row][i]=0;
            }
        }
        return false;
    }
```

Big O Notation:

In computer science, Big O notation is a way to describe the efficiency or complexity of an algorithm in terms of the input size. It provides an upper bound on the time or space complexity of an algorithm in the worst-case scenario, using a simplified representation.

For this code, which is solving the N-Queens problem using backtracking:

isSafe function checks the safety of placing a queen at a certain position on the board.

It iterates through rows and diagonals, checking for conflicts.

Its time complexity is O(N), where N is the size of the board.

solve function uses a recursive backtracking approach to find a solution for placing N queens. Its time complexity is harder to precisely represent due to the nature of backtracking, but in the worst case, it explores all possible configurations of placing queens.

Its time complexity is typically O(N^N) or exponential, as it can potentially generate and explore N^N possible configurations (where N is the size of the board).

For the N-Queens problem, the worst-case time complexity is often exponential due to the large number of possibilities to explore, making it computationally expensive for larger board sizes.

The Big O notation helps in understanding how the algorithm's performance scales with increasing input size.

In the case of the N-Queens problem, as the board size grows, the time complexity grows exponentially, making it less practical for larger N values without optimization.

4) Team members

Students of the Faculty of Computers and Artificial Intelligence at Helwan University, Department of Computer Science

Course : Operating System – 2 (fall 2023)

Name	ID
Ossama Samir	20210140
Abd-ulla Mohamed	20210556
Marowa Omar	20210900
Rana Essam	20210335
Toka Mohamed	20210242
Nader Mamdouh	202000976
Omina Abdel-latif	202000154

5) References

1) N Queens Problems - javatpoint

2)The n-queens completion problem | Research in the Mathematical Sciences (springer.com)

3) N Queen Problem - GeeksforGeeks