PLACE 8 QUEENS ON A CHESSBOARD SO THEY DON'T KILL EACH OTHER

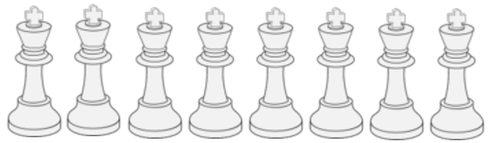
PLACE 8 QUEENS ON A CHESSBOARD SO THEY DON'T KILL EACH OTHER

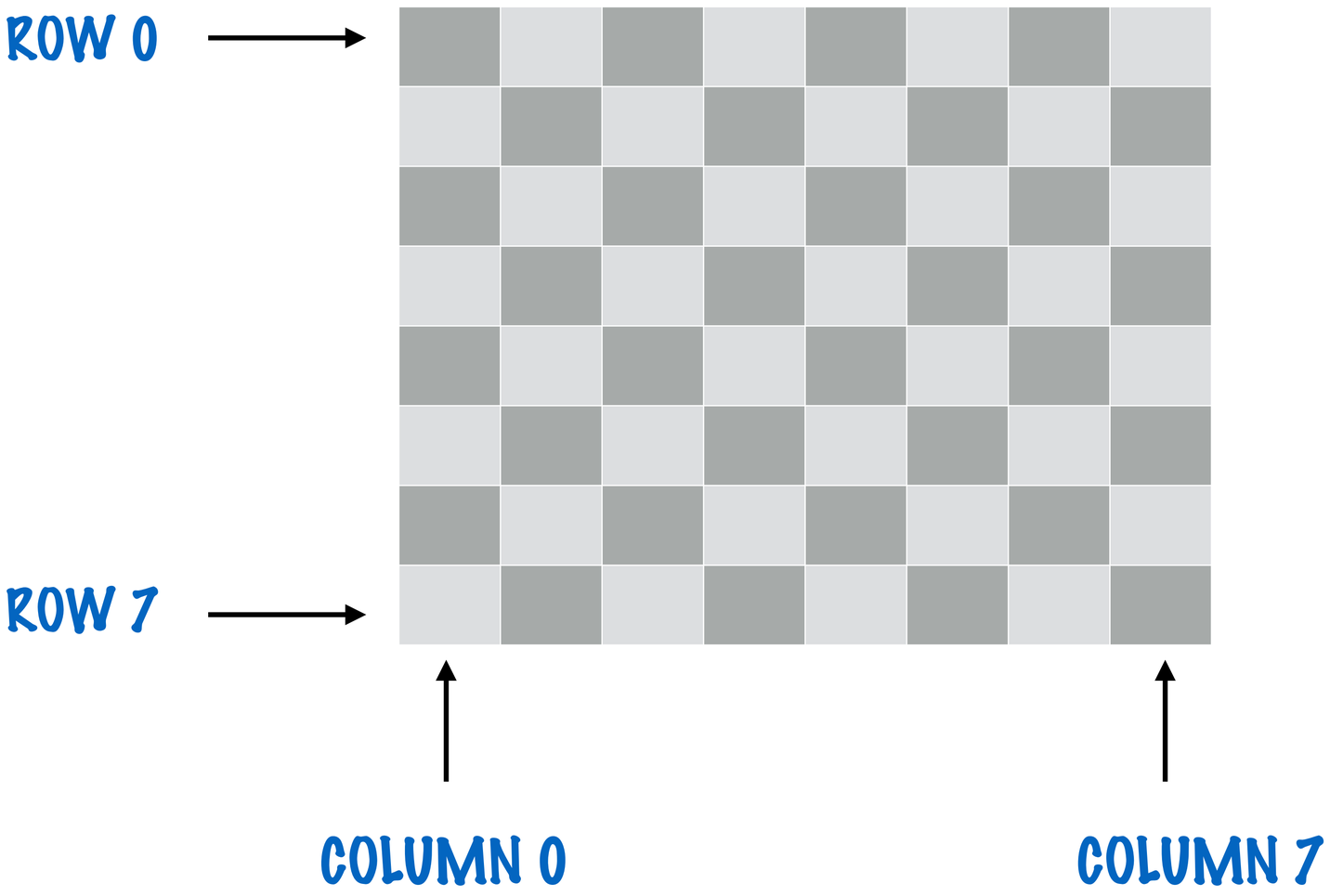
REMEMBER THE QUEENS CAN MOVE ALONG THEIR ROWS, COLUMNS AND ALONG BOTH DIAGONALS

PLACE ONE QUEEN AT A TIME, ONE ON EACH ROW OR ONE ON EACH COLUMN

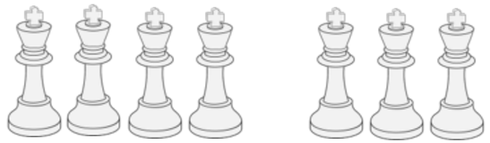
REMEMBER TO CHECK AT EVERY STEP TO SEE IF THE QUEEN PLACED IS SAFE, AND IF THE REMAINING QUEENS CAN BE PLACED WITH THAT CONFIGURATION

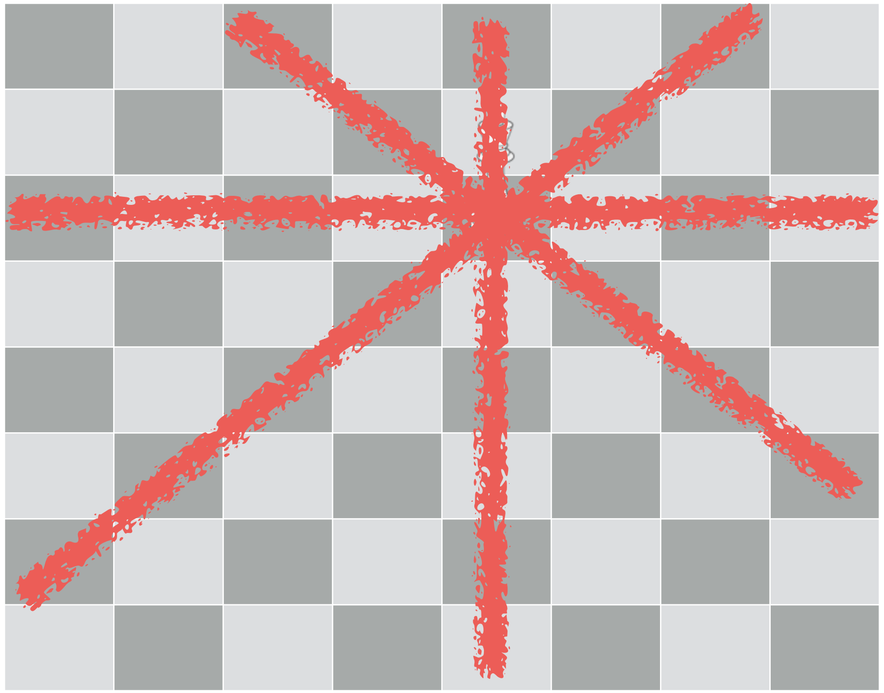
EIGHT QUEENS



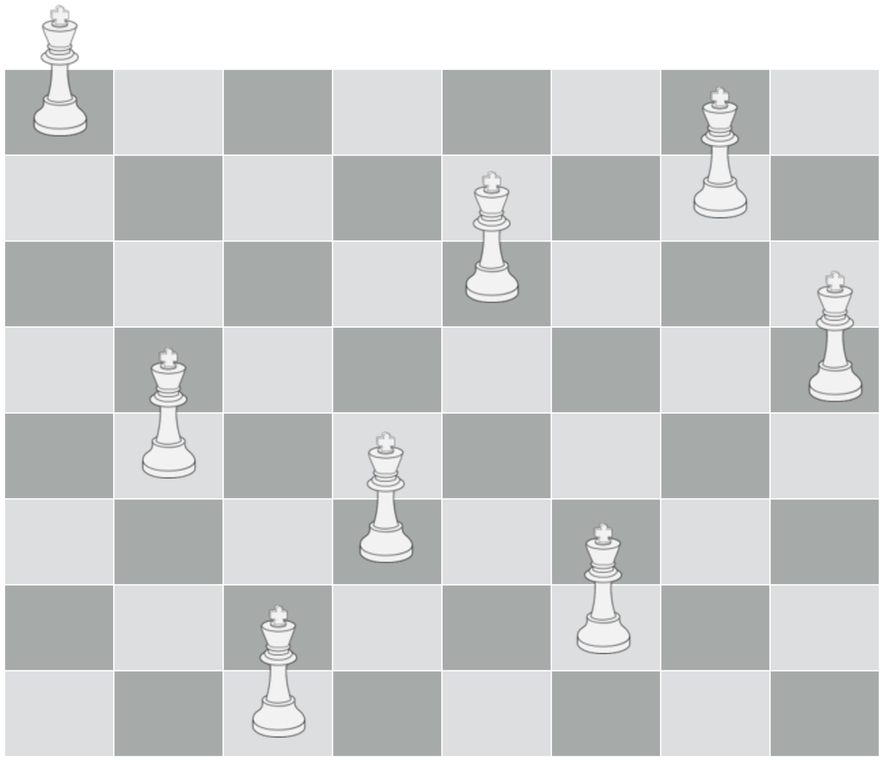


EIGHT QUEENS





EIGHT QUEENS



PLACE A QUEEN IN EACH ROW OR COLUMN IN A SAFE POSITION

SEE IF THE REMAINING QUEENS CAN BE PLACED SAFELY WITH THE CURRENT POSITION OF THE QUEEN

RECURSIVELY DO THIS TILL THE RIGHT POSITIONS FOR ALL QUEENS HAVE BEEN FOUND

WHAT IS THE BASE CASE?

1. THE QUEENS HAVE BEEN ALL PLACED AND WE'RE BEYOND THE BOUNDARY OF THE CHESSBOARD

WHAT IS THE RECURSIVE CASE?

KEEP PLACING QUEENS IN DIFFERENT POSITIONS OF THE SAME ROW OR COLUMN

CHECK WHETHER THE REMAINING QUEENS CAN BE SUCCESSFULLY PLACED

THERE ARE MANY APPROACHES TO THIS PROBLEM, WE'LL IMPLEMENT ONE APPROACH WHICH IS PRETTY INTUITIVE

FIRST WE'LL SET UP A WHOLE BUNCH OF HELPER METHODS TO CHECK WHETHER A PARTICULAR POSITION IS SAFE

HELPER METHODS FOR CHECKING THE SPECIFIED, ROW, COLUMN AS WELL AS BOTH DIAGONALS

CHECK IF THE CURRENT QUEEN POSITION IS SAFE

```java
public static boolean isSafe(int[][] chessBoard, int row, int col) {
    if (!isColumnSafe(chessBoard, col)) {
        return false;
    }
    if (!isRowSafe(chessBoard, row)) {
        return false;
    }
    if (!isLeftDiagonalSafe(chessBoard, row, col)) {
        return false;
    };

    return isRightDiagonalSafe(chessBoard, row, col);
}
```

NOTE THE CHESSBOARD IS AN INTEGER MATRIX

1 IN A CELL INDICATES THE PRESENCE OF A QUEEN

CHECK IF THE ROW, COLUMN AND NEITHER OF THE DIAGONALS HAVE A QUEEN ALREADY PRESENT

CHECK THE ROW AND THE COLUMN

```java
private static boolean isColumnSafe(int[][] chessBoard, int col) {
    int colSum = 0;
    // Check if there is another queen in the same column.
    for (int r = 0; r < N; r++) {
        colSum += chessBoard[r][col];
    }

    return colSum == 1;
}

private static boolean isRowSafe(int[][] chessBoard, int row) {
    int rowSum = 0;
    // Check if there is another queen in the same row.
    for (int c = 0; c < N; c++) {
        rowSum += chessBoard[row][c];
    }

    return rowSum == 1;
}
```

JUST SUM THE CELLS IN THE ROW OR COLUMN, IT SHOULD BE EXACTLY 1 MEANING ONE QUEEN IS PRESENT

CHECK THE LEFT DIAGONAL

```java
private static boolean isLeftDiagonalSafe(int[][] chessBoard, int row, int col) {
    // Check if there is another queen in the same left diagonal.
    int leftDiagSum = 0;
    int r = 0;
    int c = 0;
    if (row > col) {
        r = row - col;
    } else {
        c = col - row;
    }
    while (r < N && c < N) {
        leftDiagSum += chessBoard[r++][c++];
    }

    return leftDiagSum == 1;
}
```

FIND THE INITIAL POSITION TRACING THE LEFT DIAGONAL FROM THE FIRST CELL IN THE LINE OF THE PLACED QUEEN

VISIT EACH CELL IN THE DIAGONAL AND CHECK THAT THERE IS EXACTLY ONE QUEEN IN THAT DIAGONAL

CHECK THE RIGHT DIAGONAL

```
private static boolean isRightDiagonalSafe(int[][] chessBoard, int row, int col) {
    int rightDiagSum = 0;
    int r = 0;
    int c = 7;
    if (row + col < N) {
        c = Math.min(row + col, N - 1);
    } else {
        r = (row + col) % (N - 1);
    }
    while (r < N && c >= 0) {
        rightDiagSum += chessBoard[r++][c--];
    }

    return rightDiagSum == 1;
}
```

FIND THE INITIAL POSITION TRACING THE RIGHT DIAGONAL FROM THE FIRST CELL IN THE LINE OF THE PLACED QUEEN

INCREMENT ROWS AND DECREMENT COLUMNS TO TRAVERSE THE RIGHT DIAGONAL

NOW PLACE THE QUEENS

```java
public static boolean placeQueen(int[][] chessBoard, int col) {
    if (col >= N) {
        return true;
    }

    for (int row = 0; row < N; row++) {
        chessBoard[row][col] = 1;
        if (isSafe(chessBoard, row, col)) {
            if (placeQueen(chessBoard, col + 1)) {
                return true;
            }
        }
        chessBoard[row][col] = 0;
    }

    return false;
}
```

PLACE QUEENS ONE IN EACH COLUMN, IF WE GO BEYOND THE NUMBER OF COLUMNS, THAT IS THE BASE CASE - ALL QUEENS HAVE BEEN PLACED

IF THE CURRENT POSITION IS SAFE, PLACE THE QUEEN

CHECK WHETHER THE REMAINING QUEENS CAN BE PLACED, WITH THE CURRENT CONFIGURATION

RETURN TRUE IF THE QUEENS WERE SUCCESSFULLY PLACED, FALSE OTHERWISE

IF NOT, THEN REMOVE THE QUEEN FROM THE CURRENT POSITION AND TRY A NEW ONE

THE COMPLEXITY OF THIS ALGORITHM IS O(N!), WE'RE TRYING EVERY POSSIBLE POSITION FOR THE QUEENS

IT IS POSSIBLE TO IMPROVE ON THIS IN LITTLE WAYS, THERE ARE LOT'S OF SMALL MODIFICATIONS TO THIS APPROACH