ALGORITHMS AND DATA STRUCTURES II

Lecture 10

Backtracking algorithm design, Eight queens problem.

Lecturer: K. Markov

markov@u-aizu.ac.jp

BACKTRACKING

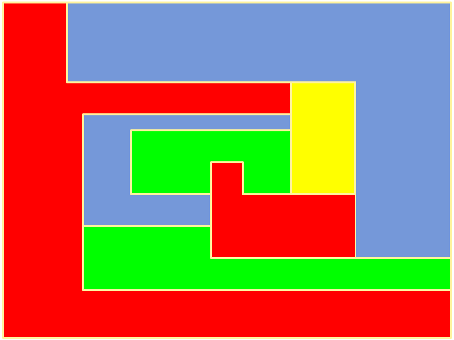
- Suppose you have to make a series of decisions, among various choices, where
  - You don't have enough information to know what to choose.
  - Each decision leads to a new set of choices.
  - Some sequence of choices (possibly more than one) may be a solution to your problem
- **Backtracking** is a methodical way of trying out various sequences of decisions, until you find one that "works"

SOLVING A MAZE

- Given a maze, find a path from start to finish
- At each intersection, you have to decide between three or fewer choices:
  - **Go straight**
  - **Go left**
  - **Go right**
- You don't have enough information to choose correctly.
- Each choice leads to another set of choices.
- One or more sequences of choices may (or may not) lead to a solution.

COLORING A MAP

- You wish to color a map with not more than four colors:
  - red, yellow, green, blue
- Adjacent countries must be in different colors.
- You don't have enough information to choose colors.
- Each choice leads to another set of choices.
- One or more sequences of choices may (or may not) lead to a solution.



BACKTRACKING

- For some problems, the only way to solve is to check all possibilities.
- Backtracking is a systematic way to go through all the possible configurations of a search space.
- We assume our solution is a vector (a(1),a(2), a(3), ..a(n)) where each element a(i) is selected from a finite ordered set S.

BACKTRACKING

- We build a **partial** solution v = (a(1), a(2),..., a(k)), extend it and test it.
- **If** the partial solution is still a candidate solution,

  proceed.

  else

  delete a(k) and try another possible choice for a(k).
- **If** possible choices of a(k) are exhausted, **backtrack** and try the next choice for a(k-1).

BACKTRACKING

- General iterative algorithm:

```
def BACKTRACK (S):
    // S – set of possible actions.
    S(1) ← S
    k = 1
    while (k > 0):
        while S(k) != emptySet:  // advance
            a(k) = an element in S(k)
            S(k) = S(k) - a(k)
            if (a(1),a(2),...,a(k)) is a solution:
                print (a(1),a(2),...,a(k))
            k = k + 1
            S(k) ← S
        k = k - 1                       // backtrack
```

BACKTRACKING

- General recursive solution:

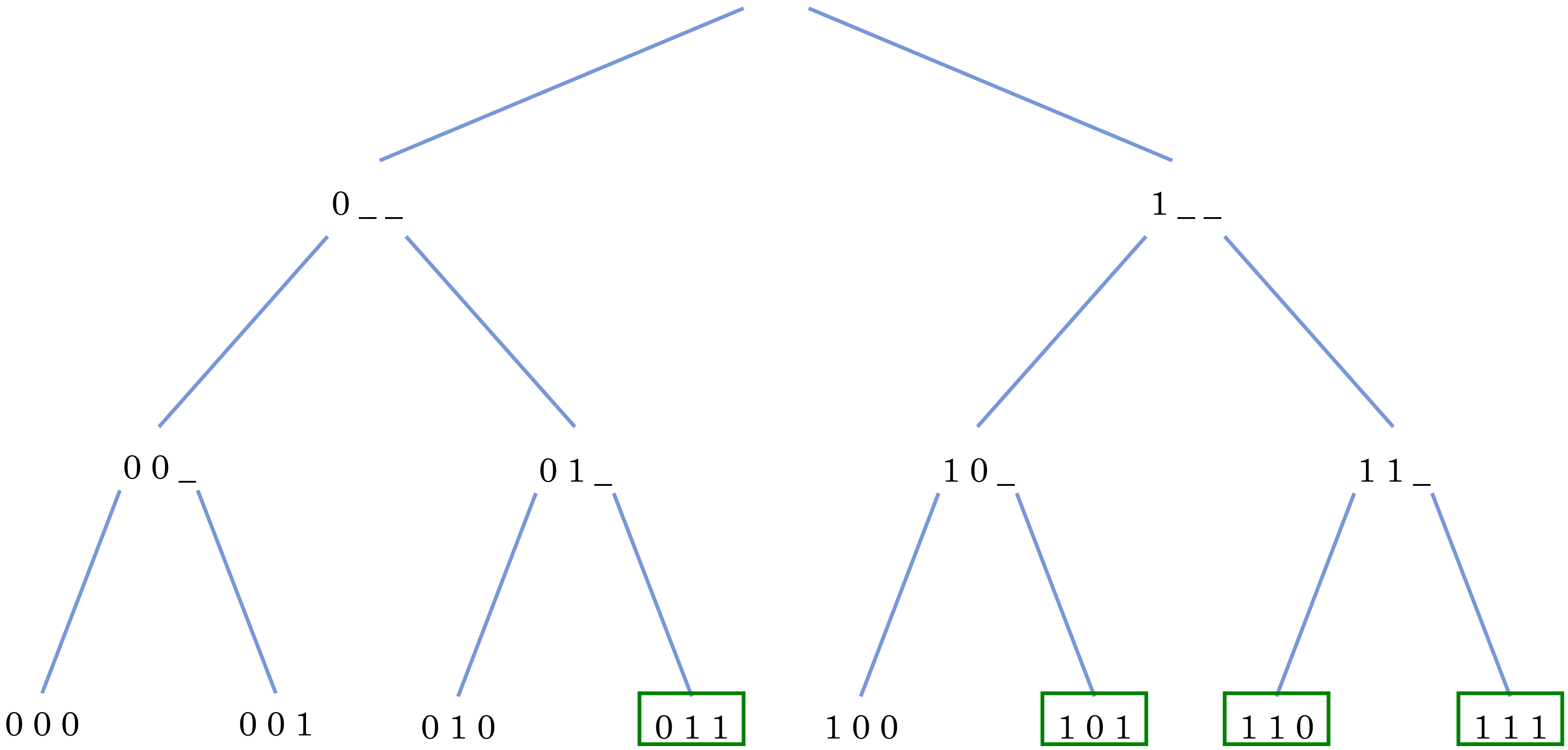
```
def BACKTRACK (A, k, S):
    // A=a(1),…,a(k) – partial solution, k – step number
    if A is a solution:
        print A= a(1),…,a(k)
    else:
        k = k +1:
        S(k) ← S
        while S(k) != emptySet:
            a(k) = an element in S(k)
            S(k) = S(k) – a(k)
            A = A + a(k)
            BACKTRACK (A, k, S)
```

BACKTRACKING

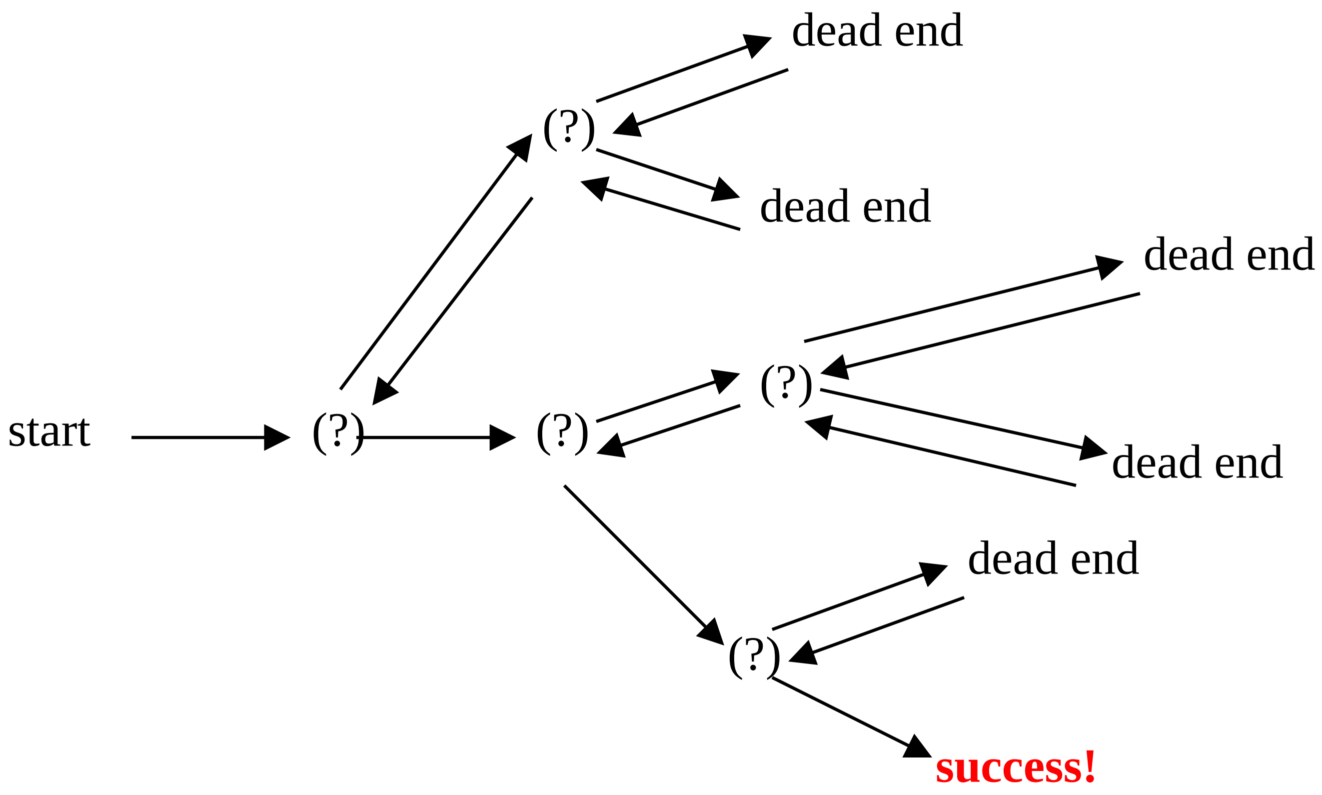
Example:

- Find out all 3-bit binary numbers for which the sum of the 1's is greater than or equal to 2.
- The only way to solve this problem is to check all the possibilities: (000, 001, 010, ...,111)
- The 8 possibilities are called the search space of the problem. They can be organized into a tree.

BACKTRACKING

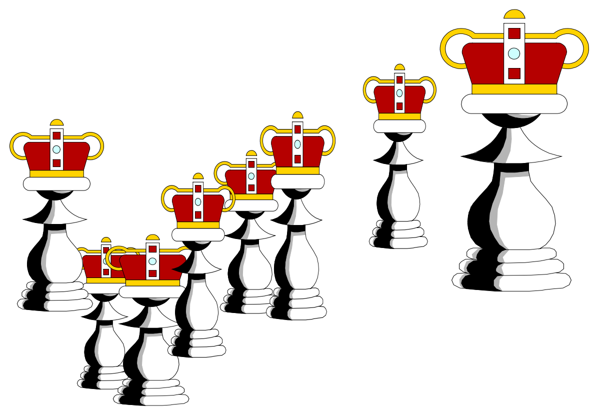


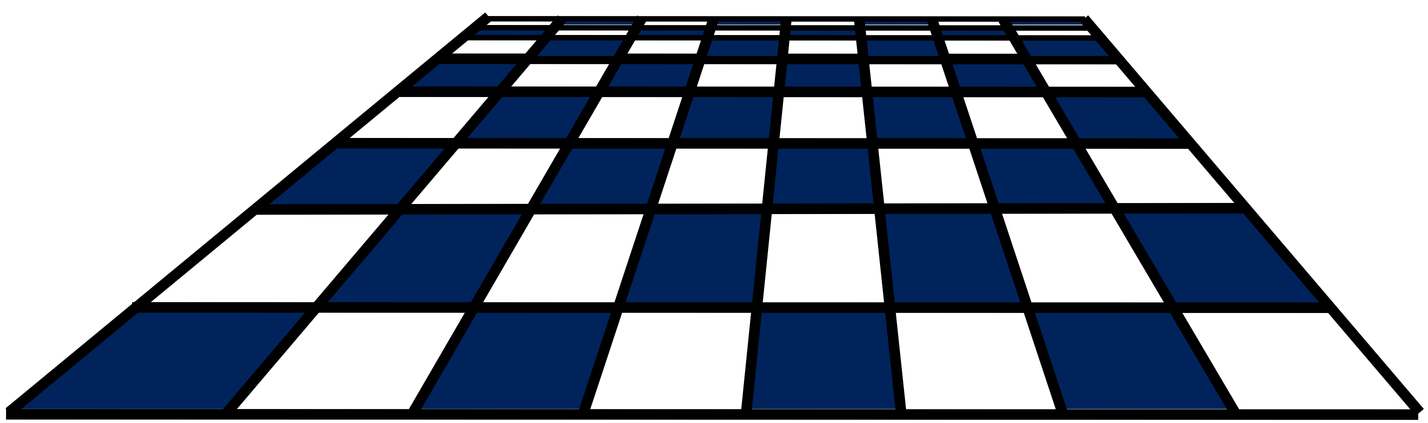
BACKTRACKING



EIGHT QUEENS PROBLEM

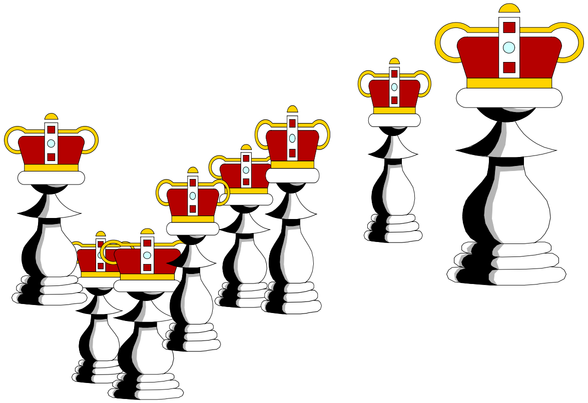
- Suppose you have 8 chess queens...
- ...and a chess board.

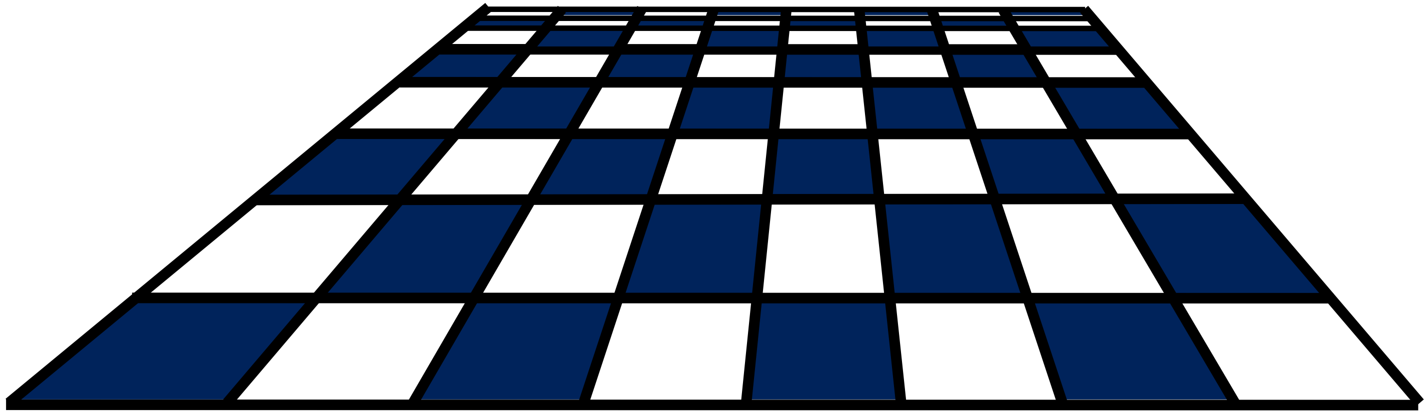




EIGHT QUEENS PROBLEM

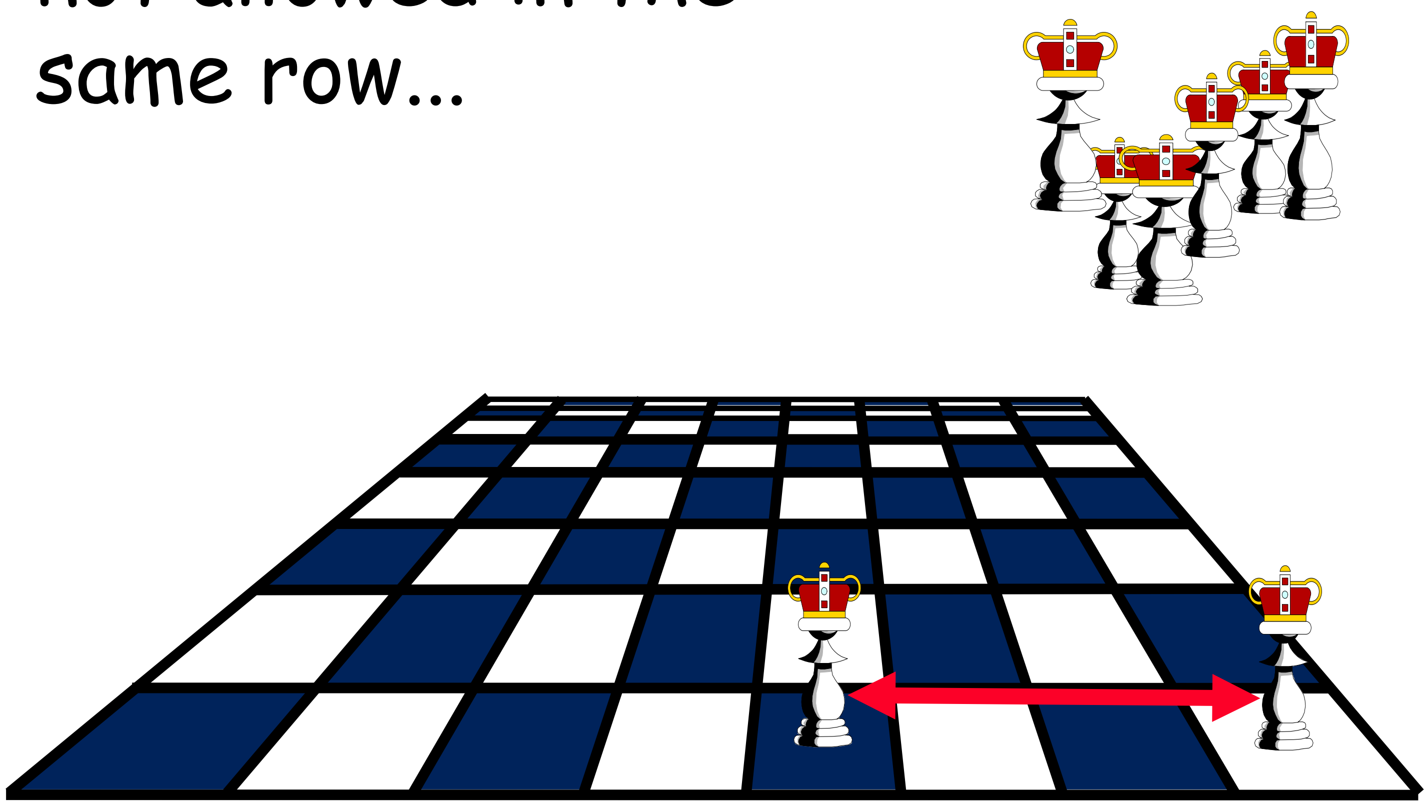
Can the queens be placed on the board so that **no** two queens are attacking each other?





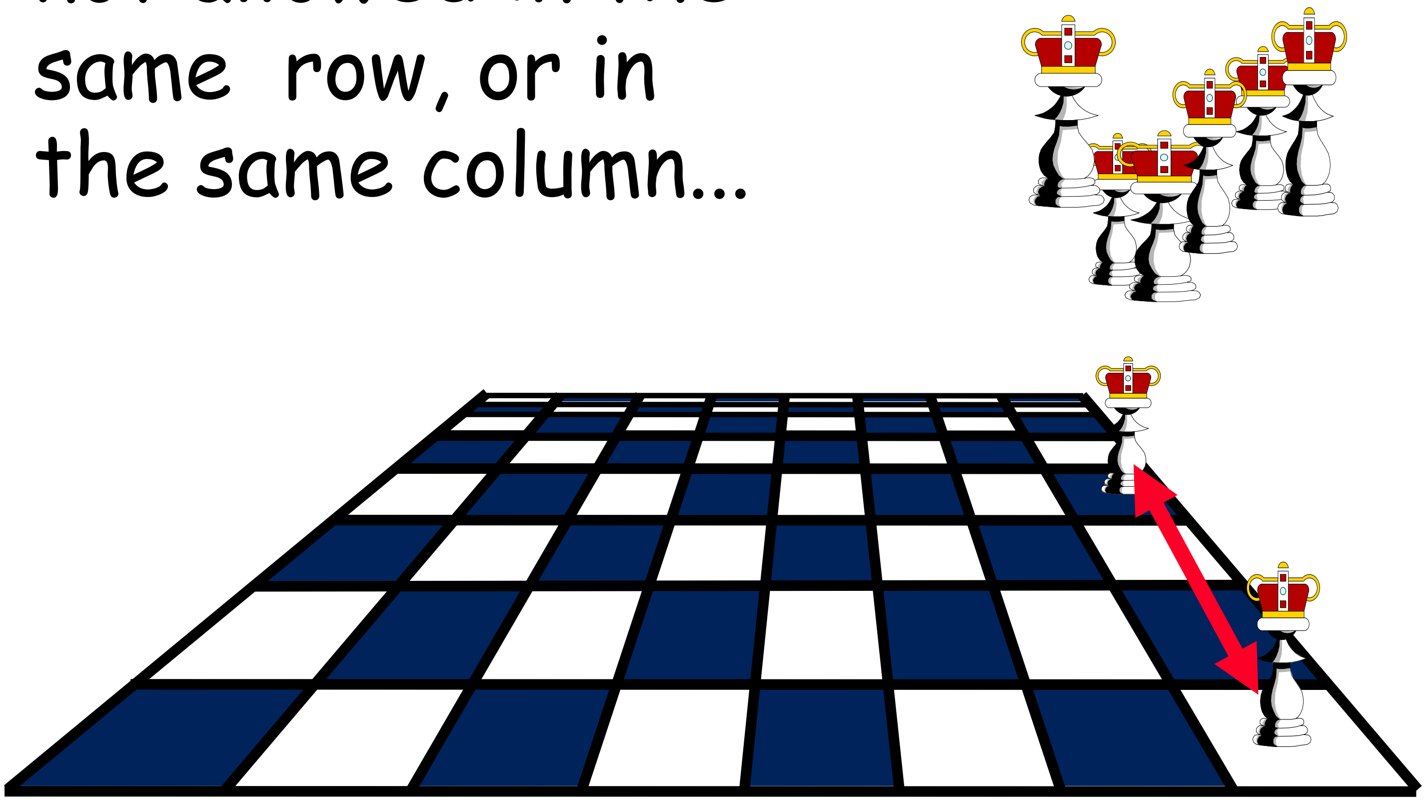
EIGHT QUEENS PROBLEM

Two queens are not allowed in the same row...



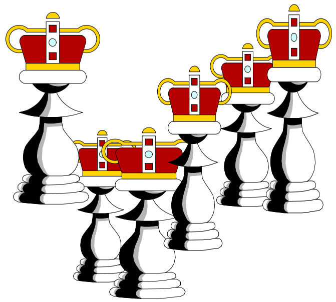
EIGHT QUEENS PROBLEM

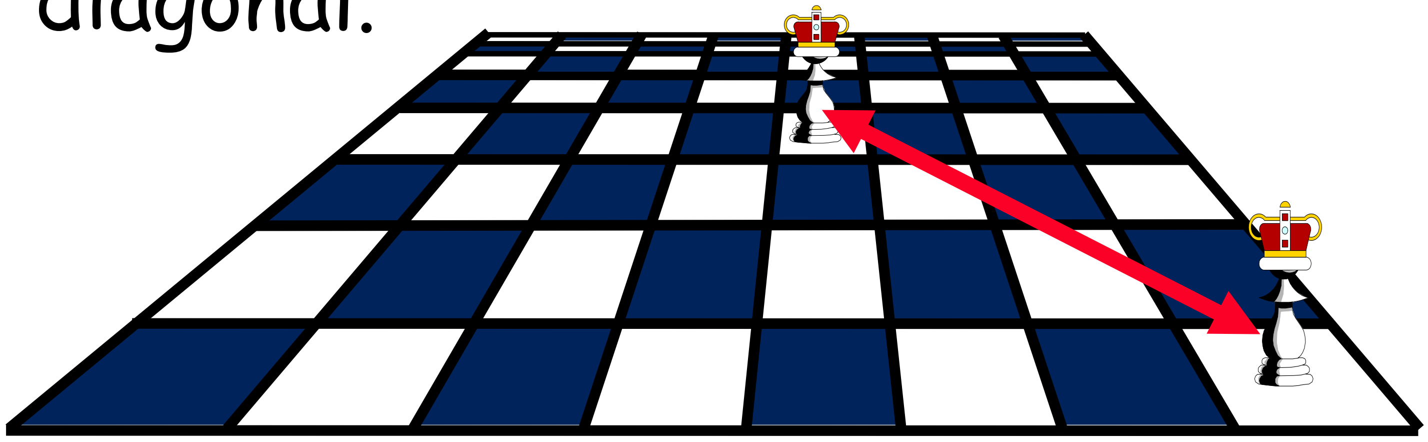
Two queens are not allowed in the same row, or in the same column...



EIGHT QUEENS PROBLEM

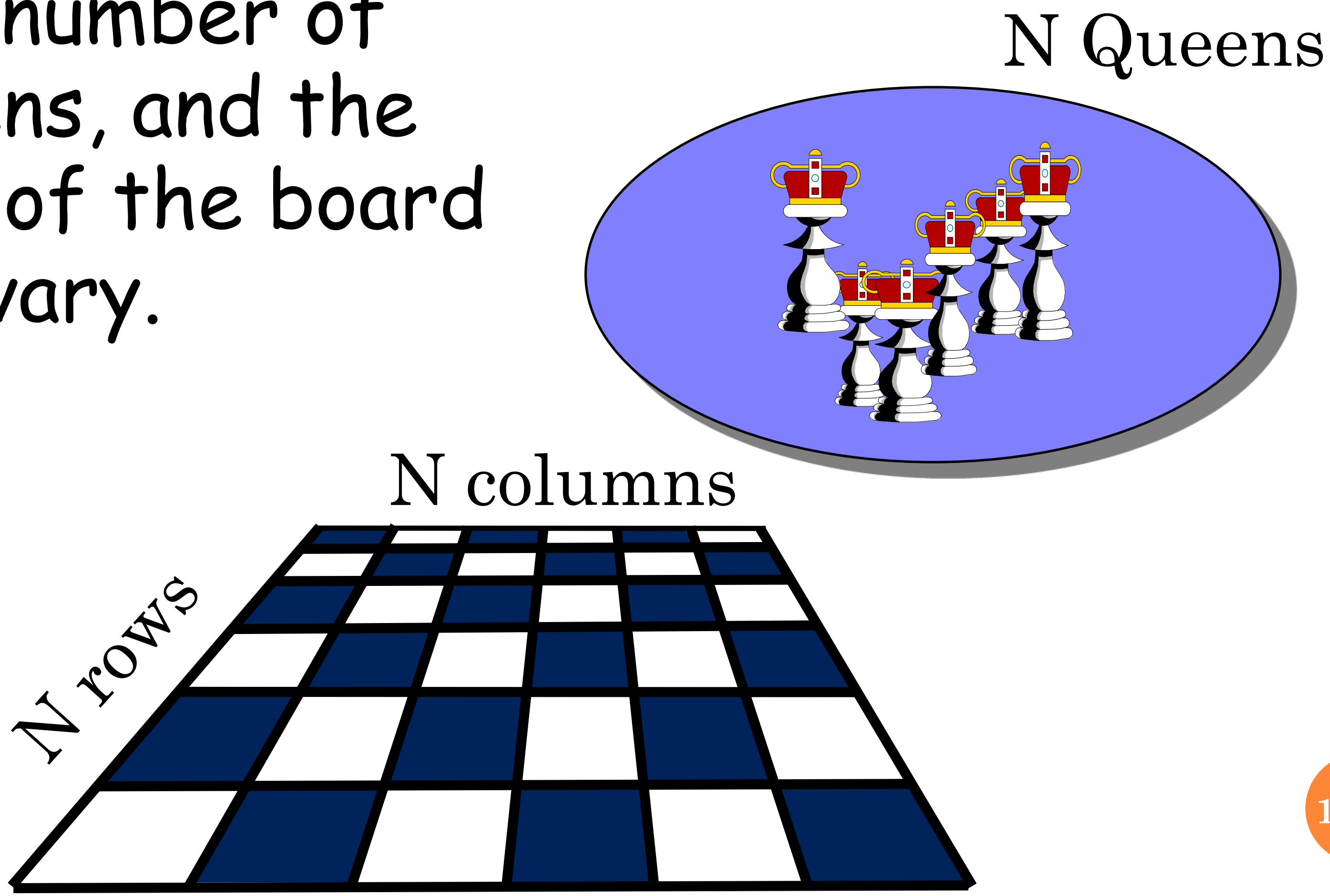
Two queens are not allowed in the same row, or in the same column, or along the same diagonal.





EIGHT QUEENS PROBLEM

The number of queens, and the size of the board can vary.



EIGHT QUEENS PROBLEM

- Brute force approach:

The problem can be solved by placing eight queens on every eight positions on a chess-board, checking each time if a solution has been obtained. This approach is of no practical use, since the number of possible positions we have to check would be

$$64^8 = 4{,}426{,}165{,}368$$

EIGHT QUEENS PROBLEM

- Can we do better?

- First, we know that two queens can not be in the same row. So, eight queens should be put in eight rows, one queen in one row. Since each queen has 8 positions to be put in the row, there are:

$$8^8 = 16{,}777{,}216 \text{ positions.}$$

EIGHT QUEENS PROBLEM

- Can we do EVEN better?

- Similarly, two queens can not be in the same column. Thus, if the queen in the first row has been put in the i-th column, the other queens can not be in the i-th column, i.e. 8 choices for the 1^{st} row, 7 choices for the 2^{nd} row,…, 1 choice for the 8^{th} row. From this, we can reduce the possible positions to

$$8! = 40,320$$

EIGHT QUEENS PROBLEM

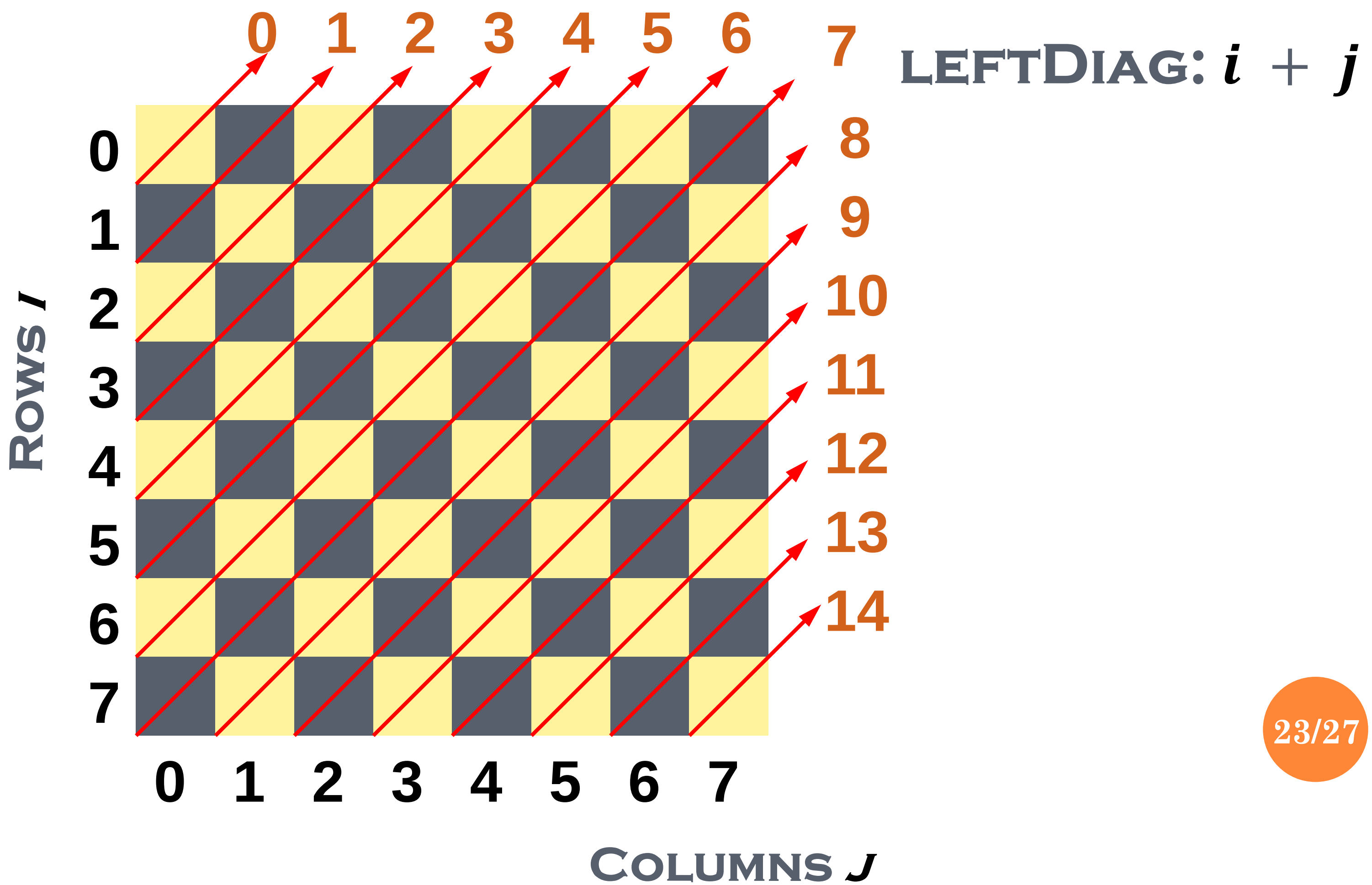
- Backtracking allows us to do much better than the above. Using a recursive call, we can realize a backtracking algorithm for eight queens problem as follows:
  - ✓ We put the queen of the first row at any position of the row.
  - ✓ Then we put the queen of the second row to a position of the row that is not attacked by the queen of the first row.

EIGHT QUEENS PROBLEM

- ✓ Assume, we have put i queens in the first i rows such that none of them attacks any of others.
- ✓ We put the queen of the $(i+1)$th row to a position of the row that is not attacked by any of the previous i queens.
- ✓ If we can not find such a position for the queen of the $(i+1)$th row, we go **BACK** to the i row to find another non-attacked position for the queen of the i row (if no such position exists we go **BACK** further to $(i-1)$th row) and then try again.

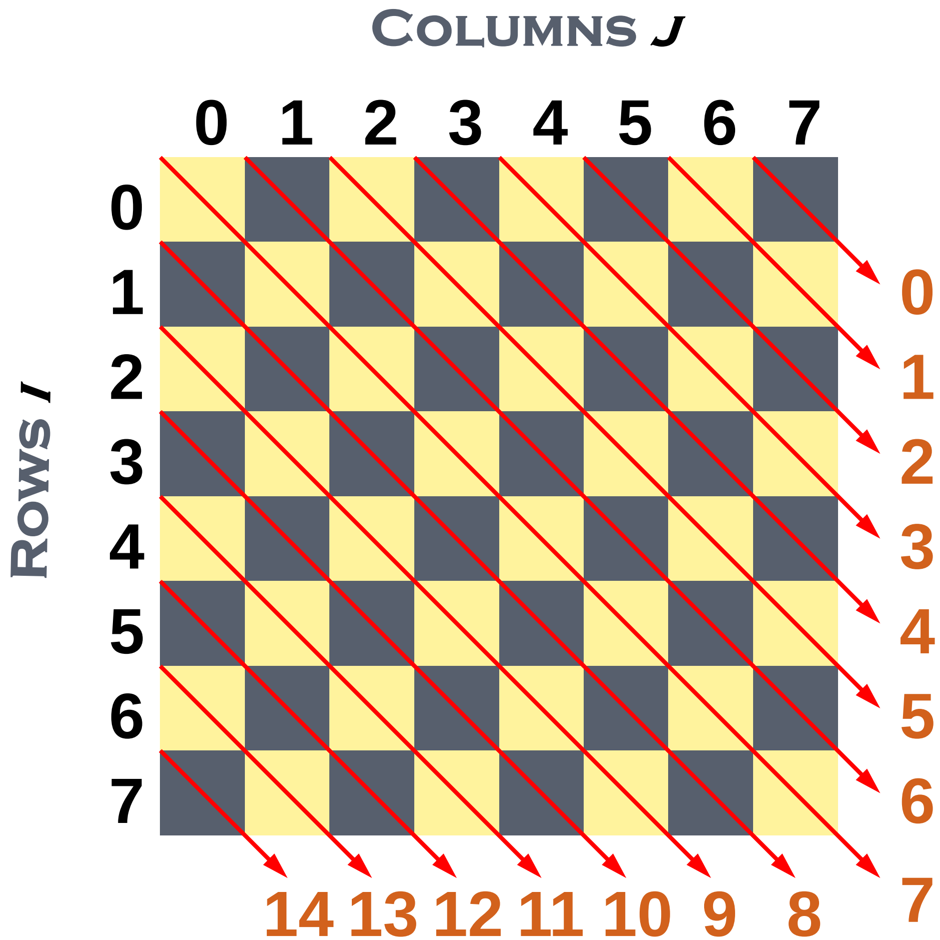
EIGHT QUEENS PROBLEM

- Implementation details – rows, columns, diagonals definition.



EIGHT QUEENS PROBLEM

- Implementation details – rows, columns, diagonals definition.



RIGHTDIAG:

$$i - j + (n - 1)$$

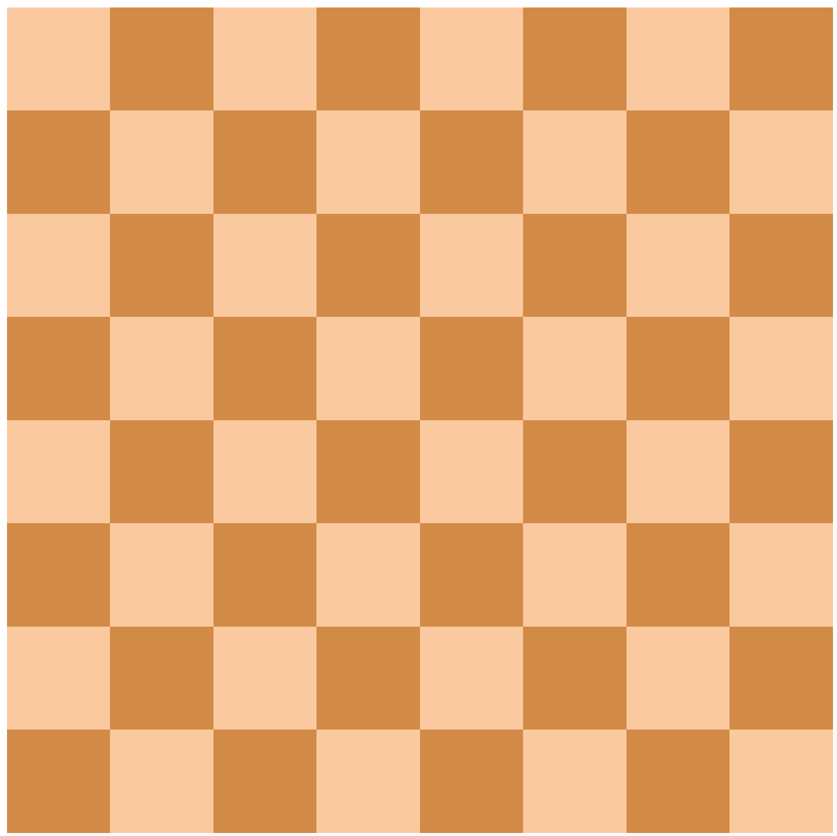
EIGHT QUEENS PROBLEM

- Pseudo-code:

```
def PutQueen(row):
    for col = 0 to n
        if column[col] = available and leftDiag[row+col] =
                available and rightDiag[row-col+n-1] = available:
            positionInRow[row] = col
            column[col] = not available
            leftDiag[row+col] = not available
            rightDiag[row-col+n-1] = not available
            if row < n-1:
                PutQueen (row+1)
            else:
                print "solution found"
        column[col] = available
        leftDiag[row+col] = available
        rightDiag[row-col+n-1]= available
```

EIGHT QUEENS PROBLEM

- Animated example:



That's All for Today!