Backtracking

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I

Preliminaries

- Exhaustive Search
  - What is an exhaustive search?
    - a trivial but very general problem-solving technique that consists of:
      1) systematically enumerating all possible **candidates** for the solution, and
      2) checking whether each candidate satisfies the problem's statement
    - aka “brute-force” search
  - Brute-force is simple to implement
  - And given enough time, it will always find a solution

Preliminaries

- Exhaustive Search
  - Example:
    - Let's say you want to find all the possible divisors of some natural number, n.
    - The exhaustive, brute-force approach would be to enumerate ALL integers from 1 to n
    - and then check whether each of them divides n without any remainder
  - Think of brute-force as searching without a brain
  - Example 2:
    - A brute-force search of a node in a BST would ignore the ordering property and, instead, would search EACH and every single node

Preliminaries

- Exhaustive Search
  - Benefit of Brute-Force:
    - You are guaranteed to find a solution
    - Since your algorithm will ultimately try EACH AND EVERY possible **candidate** solution, you will find the real solution
  - Negative of Brute-Force:
    - It takes a LOOOOOOOONG time.
    - Sure, your algorithm, in theory, will produce a solution
    - But most likely not in your lifetime!
    - Even for average size values of n, the running time is often computationally prohibitive.

Backtracking

- What is backtracking?
  - Often no more than a clever implementation of an exhaustive search
  - BUT, the savings over a brute force algorithm can be significant
  - Backtracking could degenerate, in a worst case, to a brute force, exhaustive search
  - But in most cases, better cases, **backtracking only checks a subset of possibilities within the search**.

Backtracking

- Simple example:
  - Arrange furniture in the house
  - An exhaustive search:
    - Would find ALL POSSIBLE furniture arrangements and check to see if one fits in the house
    - This is WAAAAAAY crazy
      - Computationally prohibitive!!!
  - Backtracking:
    - Place one piece of furniture in the room
    - Then try the second and the third, and so on
    - If they all fit, then great
    - If not, remove the last piece, and continue trying

Backtracking

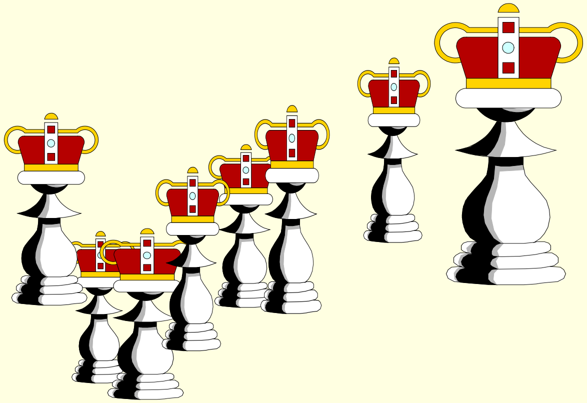
- Simple example:
  - Arrange furniture in the house
  - Backtracking:
    - In a worst case, this could result in another undo, and then another, and so forth.
      - And we could end up trying all possibilities
    - But realistically, we will terminate before then with a satisfactory arrangement
    - So we can call this a “smart brute force”
      - We try arrangements in a smart way
      - And we could possibly, in the worst case, have to check all possible arrangements

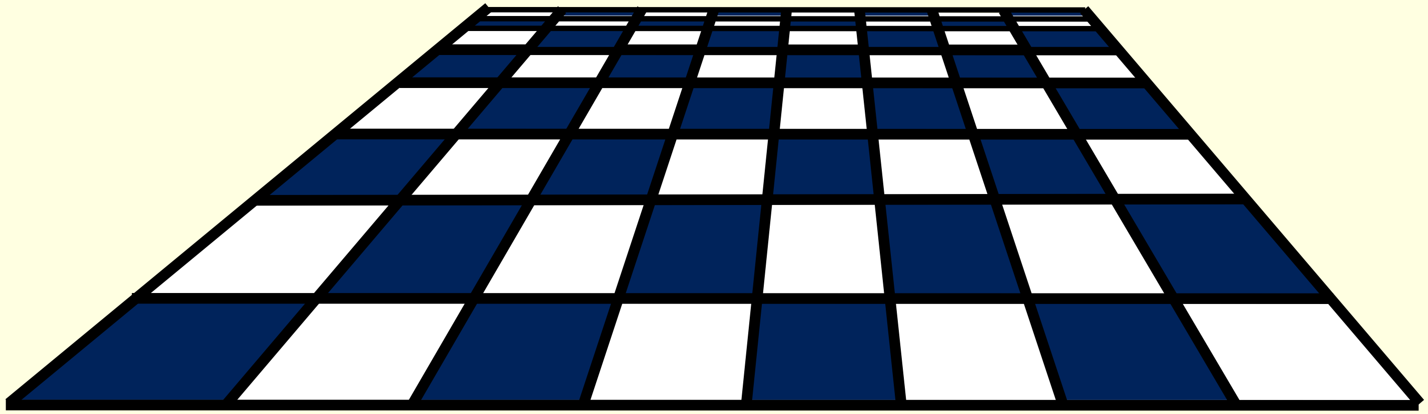
Backtracking

- Simple example:
  - Arrange furniture in the house
  - Backtracking:
    - But notice:
      - Not all arrangements are made.
      - Sofas are never attempted to be placed in the kitchen, for example
      - Other bad arrangements are discarded
    - This elimination of bad arrangements, from the outset, is known as PRUNING.
    - We prune the search space, resulting in less possibilities to check.
  - …another example…

The N-Queens Problem

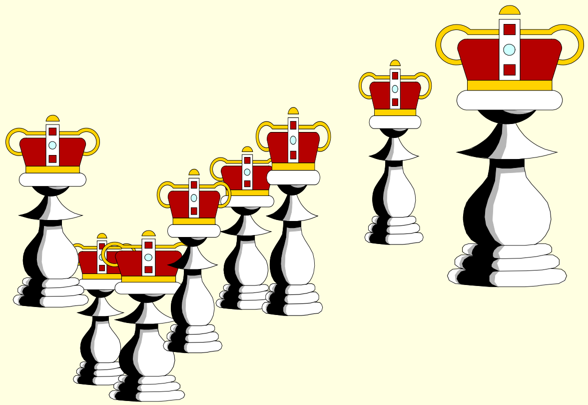
- Suppose you have 8 chess queens...
- ...and a chess board

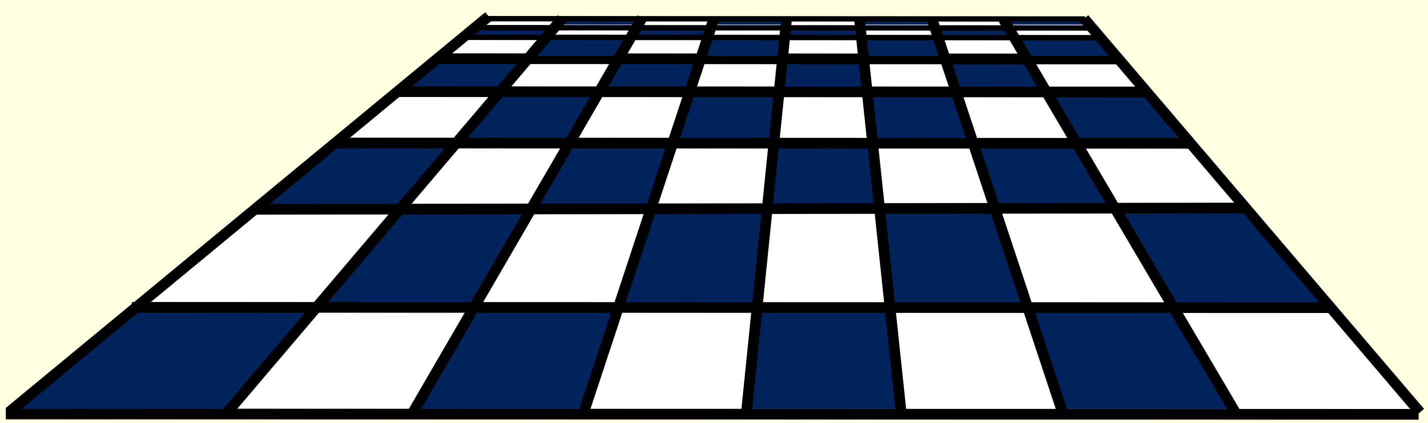




The N-Queens Problem

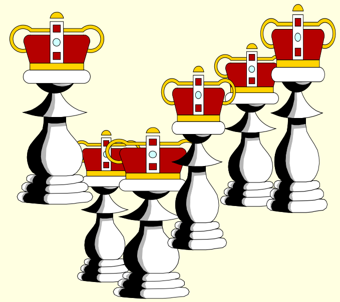
Can the queens be placed on the board so that no two queens are attacking each other?

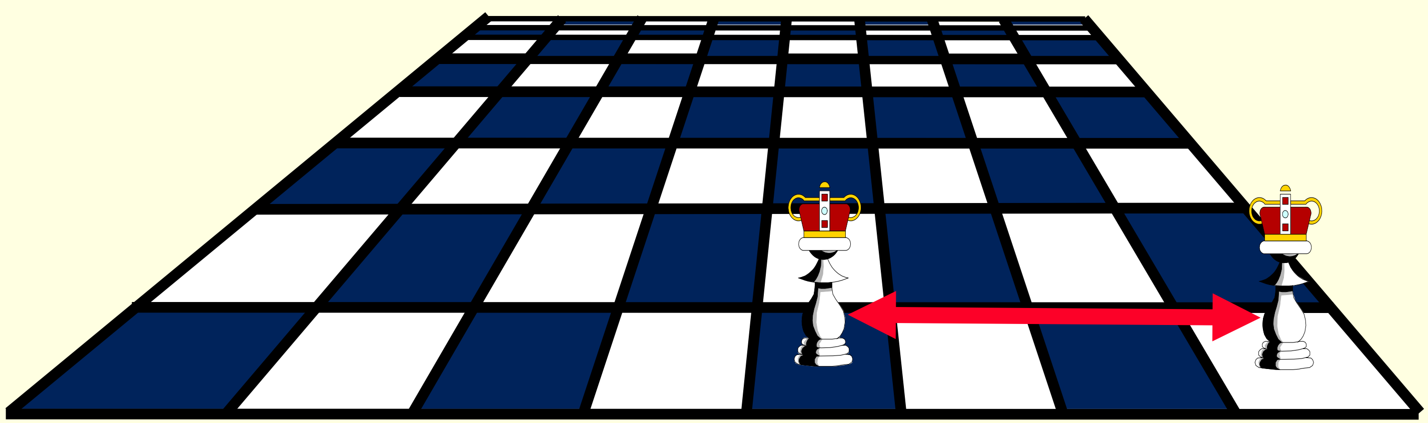




The N-Queens Problem

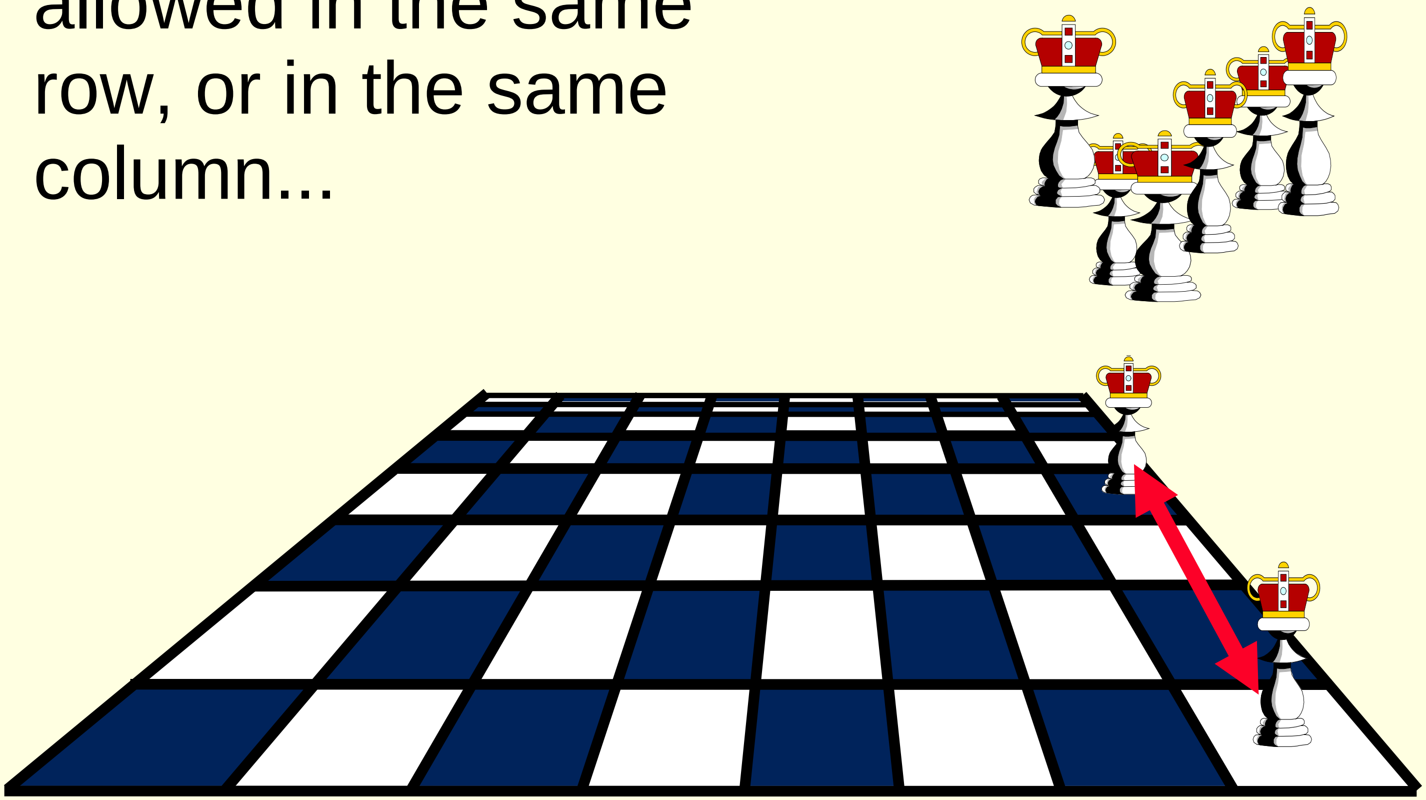
Two queens are not allowed in the same row...





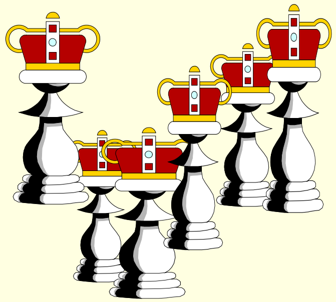
The N-Queens Problem

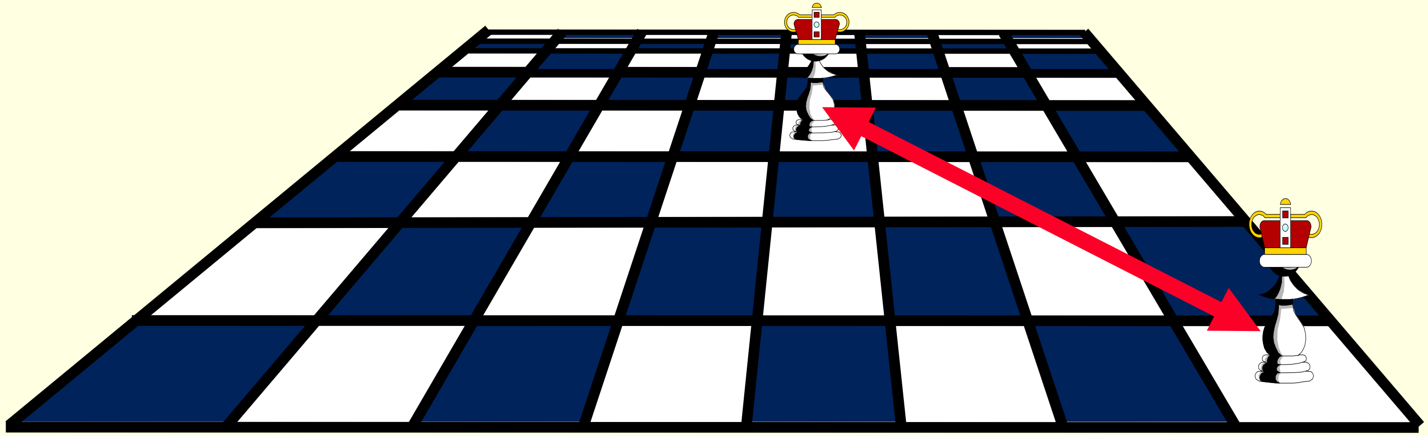
Two queens are not allowed in the same row, or in the same column...



The N-Queens Problem

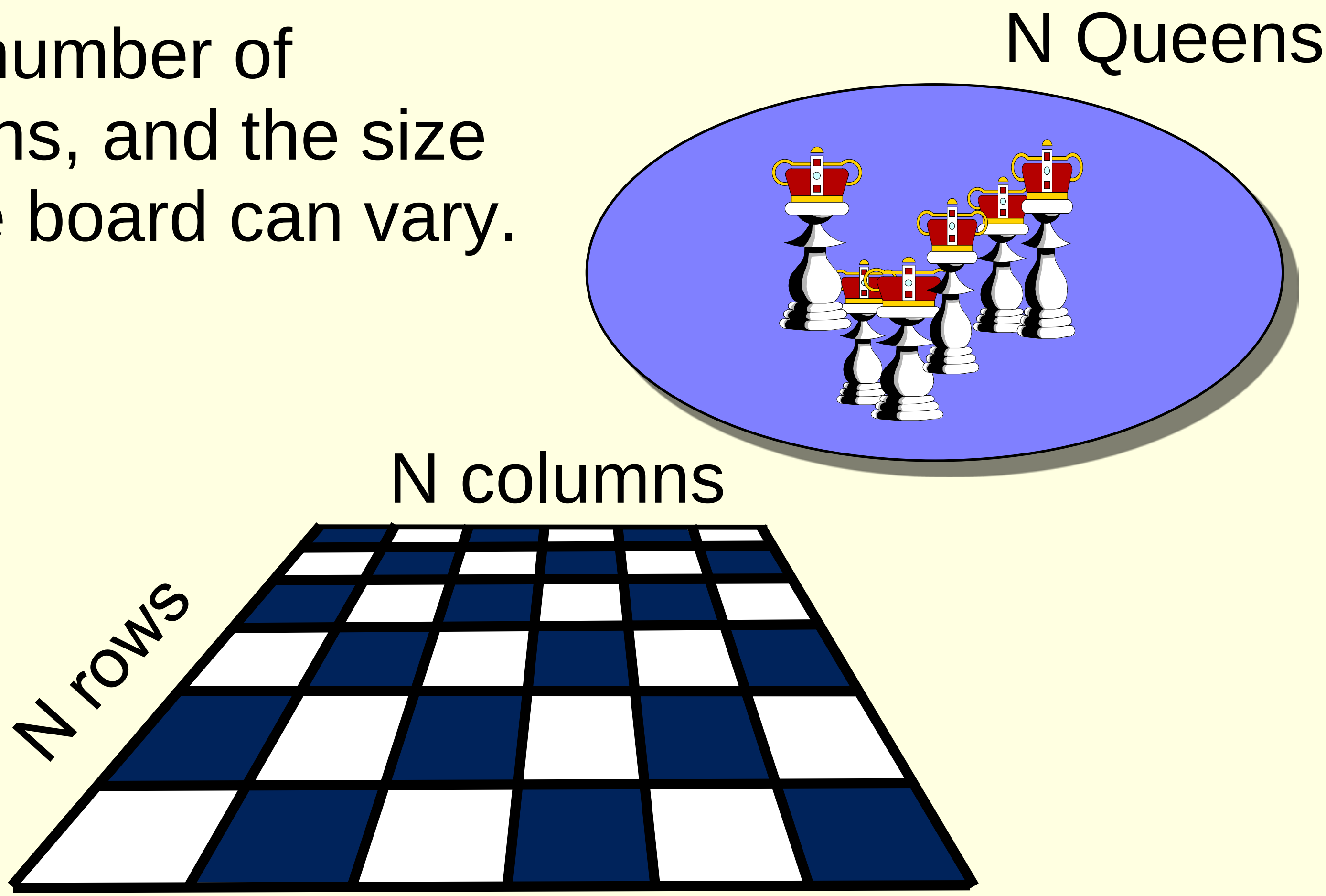
Two queens are not allowed in the same row, or in the same column, or along the same diagonal.





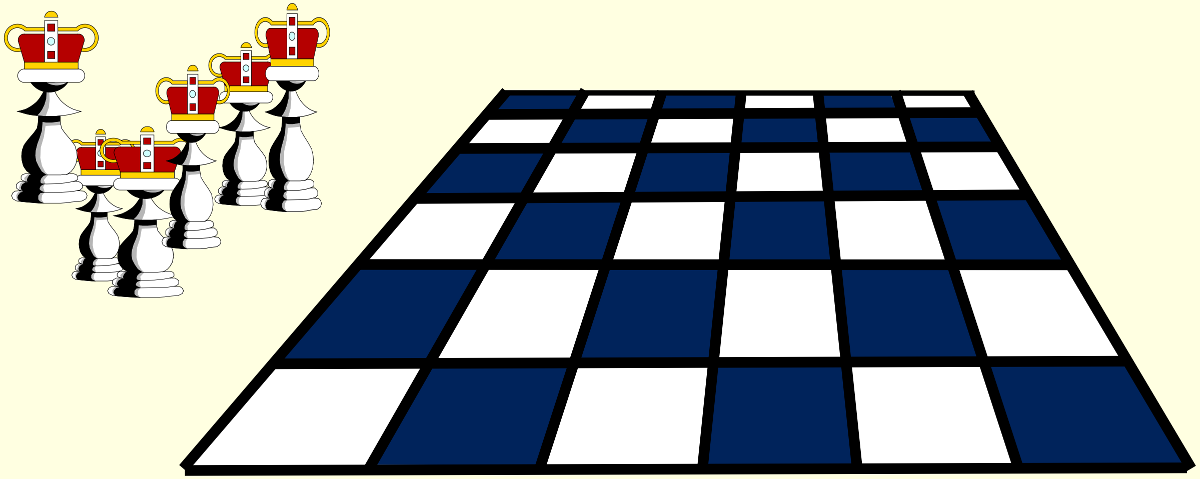
The N-Queens Problem

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



The N-Queens Problem

We will write a program which tries to find a way to place N queens on an N x N chess board.

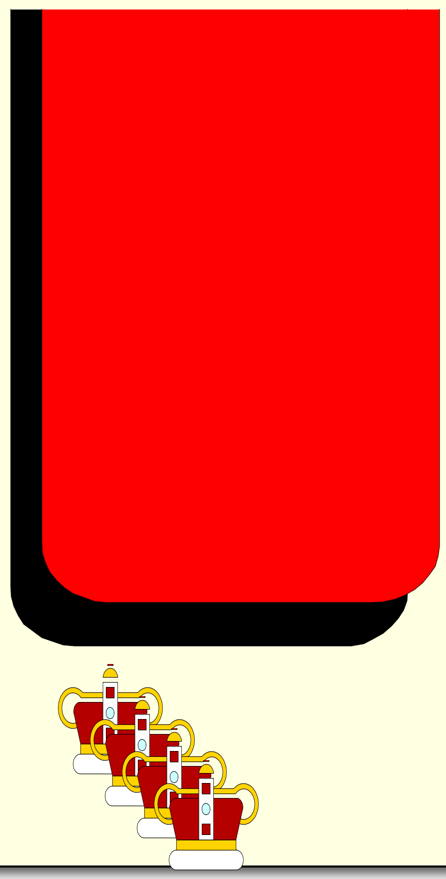


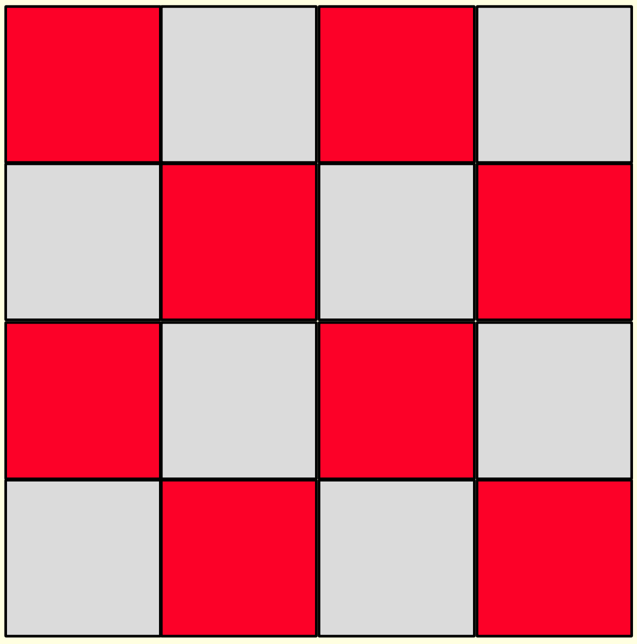
Backtracking

- N-Queens Problem:
  - So how would we do this?
  - These slides are about backtracking, so the answer is obvious. But for now, you don't know what this means exactly.
  - So what would you do?
  - Exhaustive brute force approach:
    - Find all possible arrangements of queens
      - 4,426,165,368 possible arrangements of 8 queens
    - See which ones are legal
    - Your CPU will cry…really, it will actually cry.

How the program works

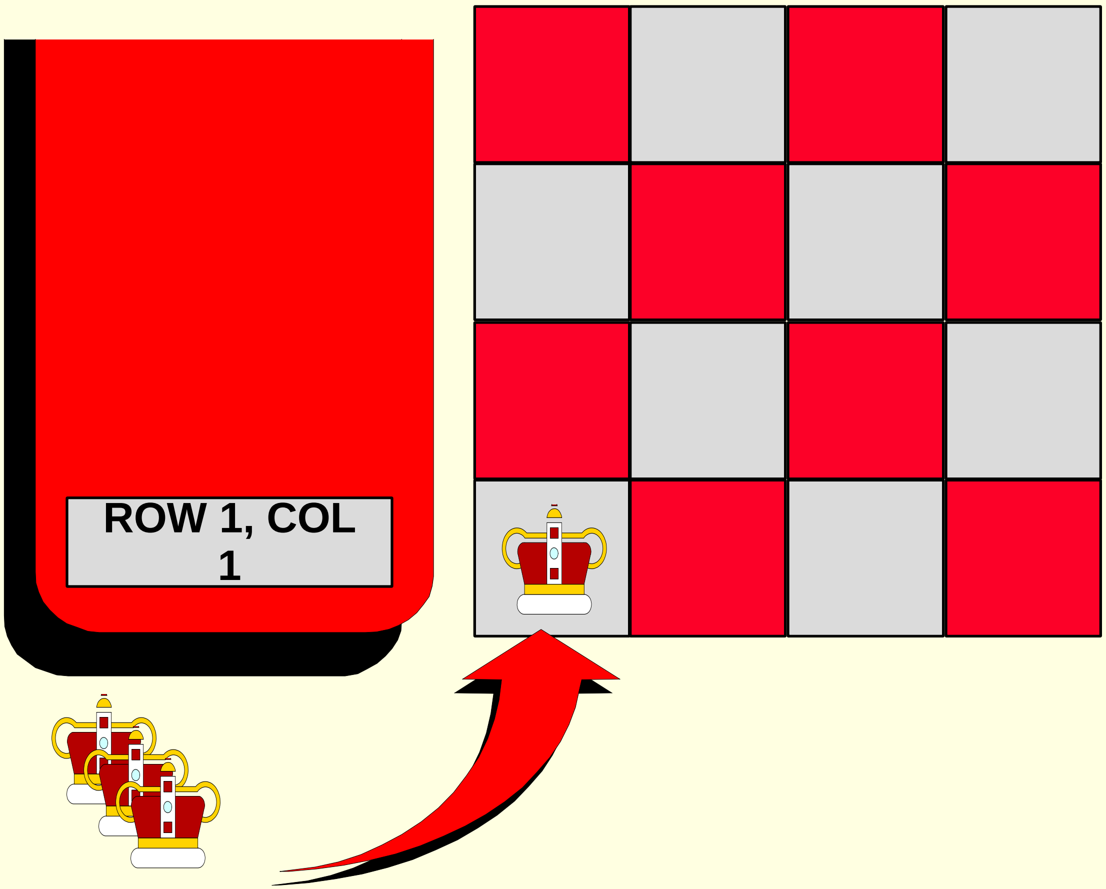
The program uses a stack to keep track of where each queen is placed.





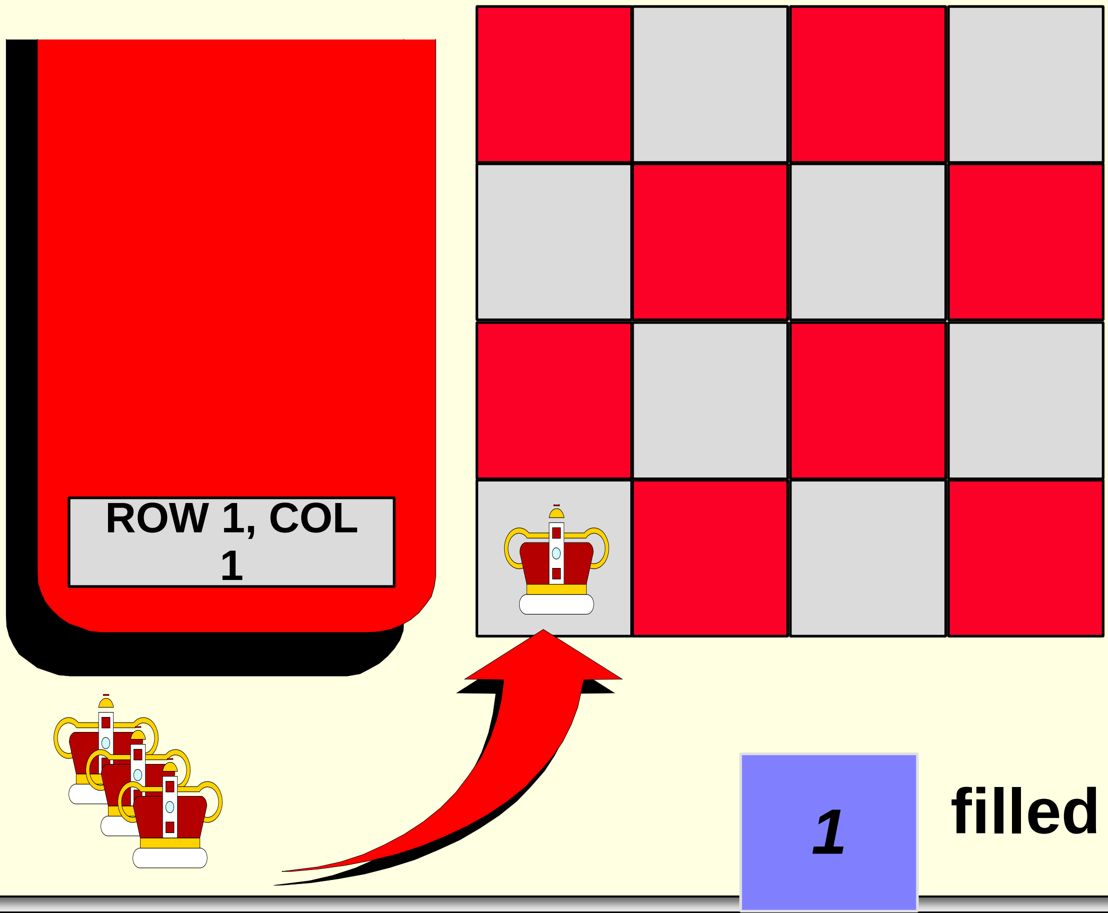
How the program works

Each time the program decides to place a queen on the board, the position of the new queen is stored in a record which is placed in the stack.



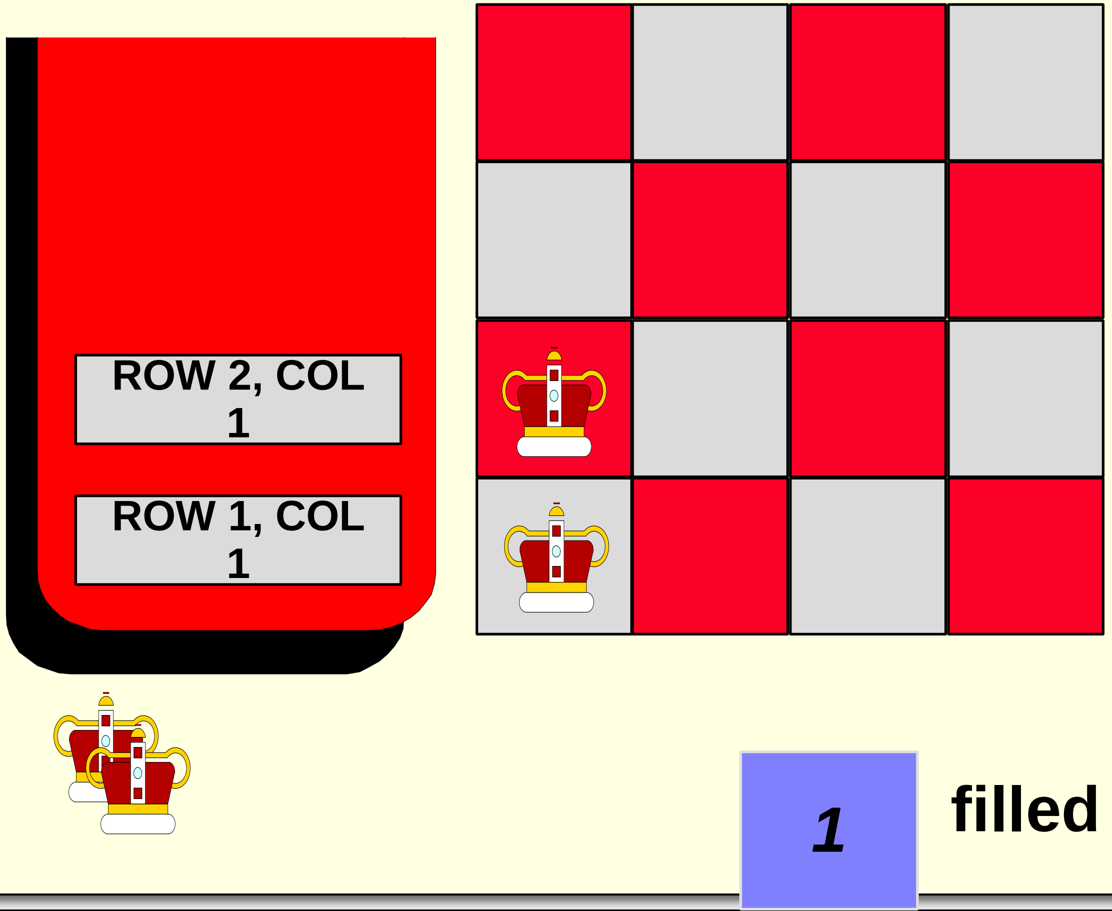
How the program works

We also have an integer variable to keep track of how many rows have been filled so far.



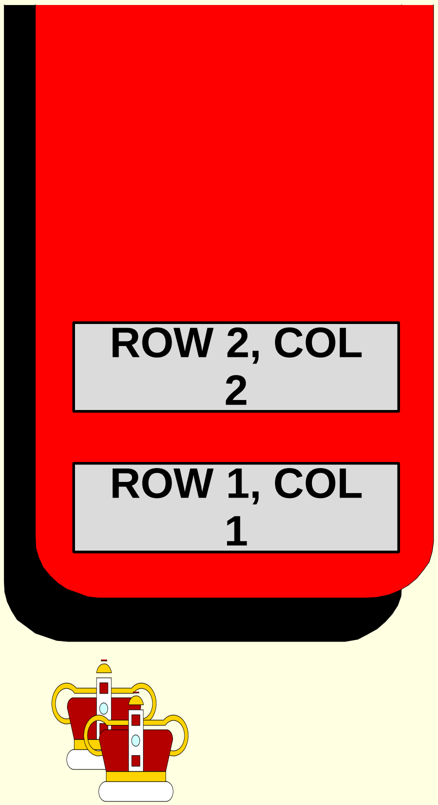
How the program works

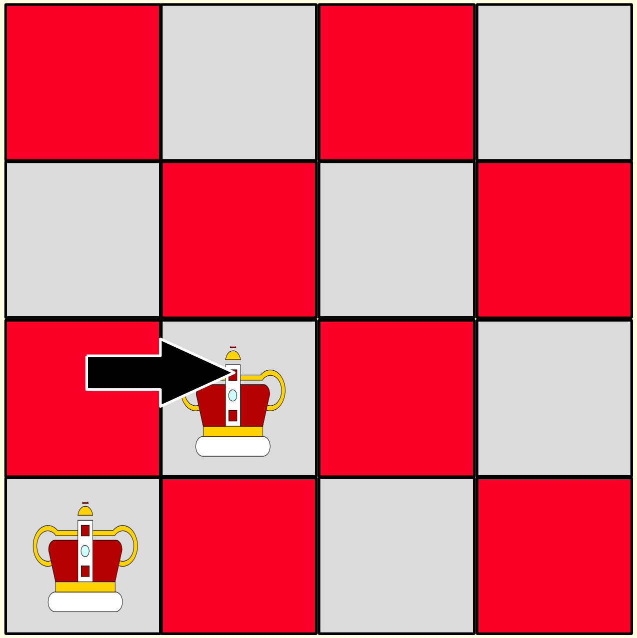
Each time we try to place a new queen in the next row, we start by placing the queen in the first column...



How the program works

...if there is a conflict with another queen, then we shift the new queen to the next column.

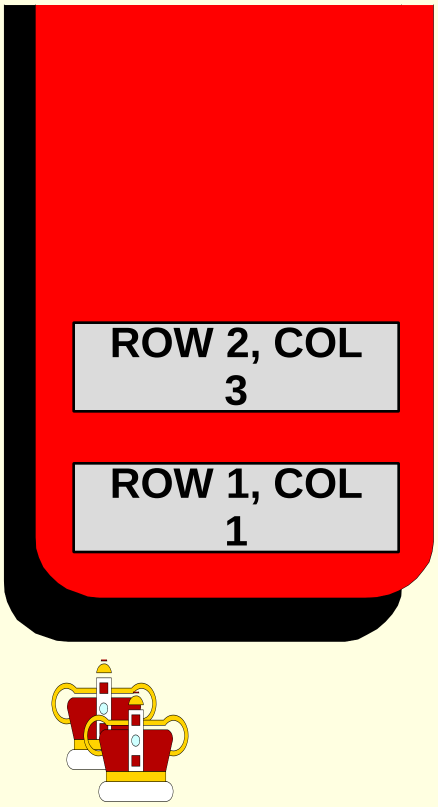


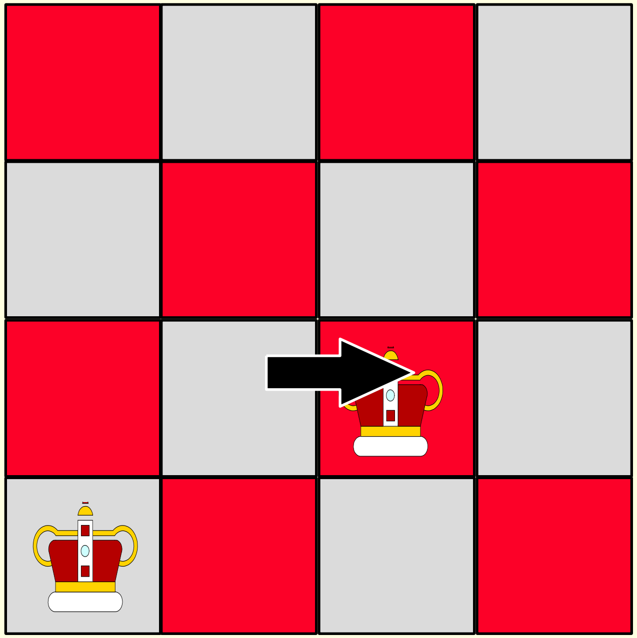


1 **filled**

How the program works

If another conflict occurs, the queen is shifted rightward again.

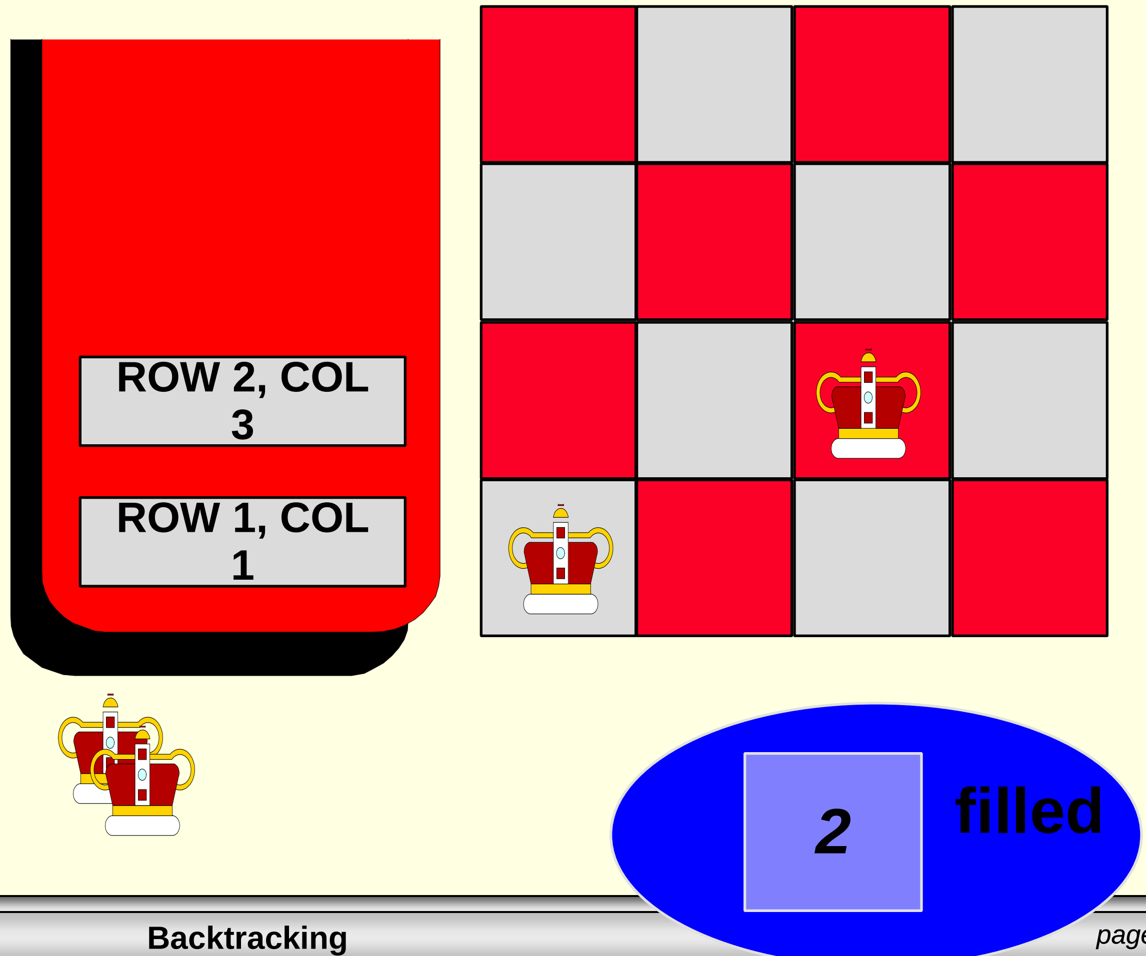




1 filled

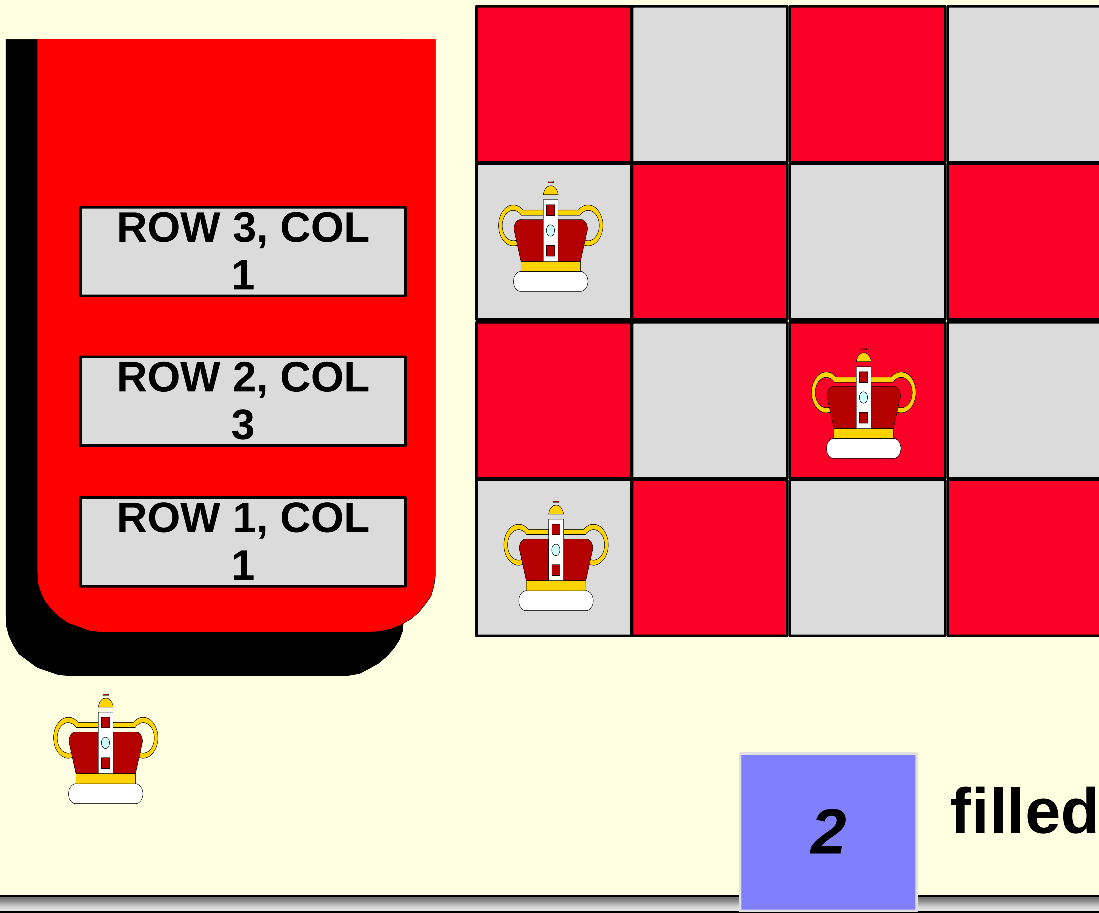
How the program works

When there are no conflicts, we stop and add one to the value of filled.



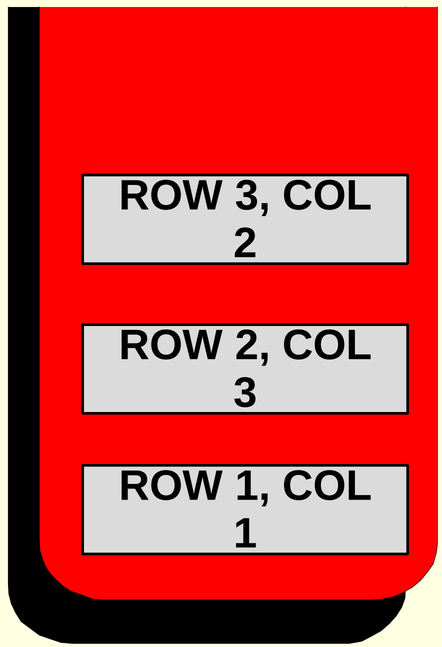
How the program works

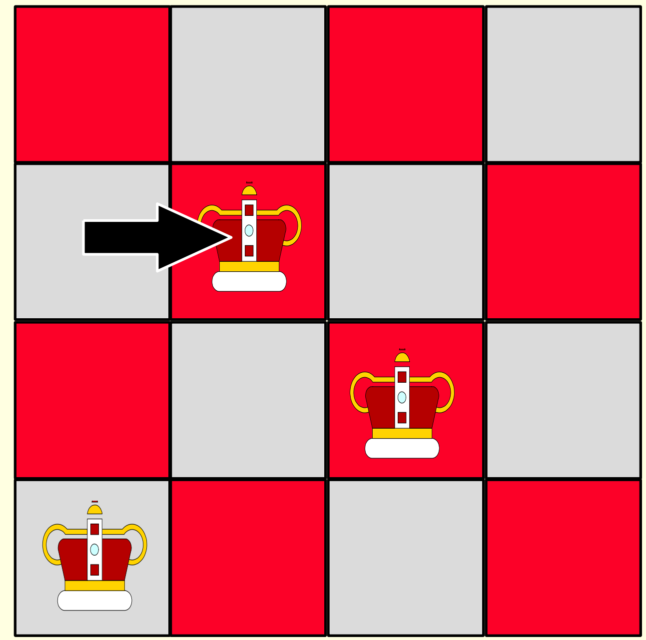
Let's look at the third row. The first position we try has a conflict...

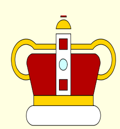


How the program works

...so we shift to column 2. But another conflict arises...





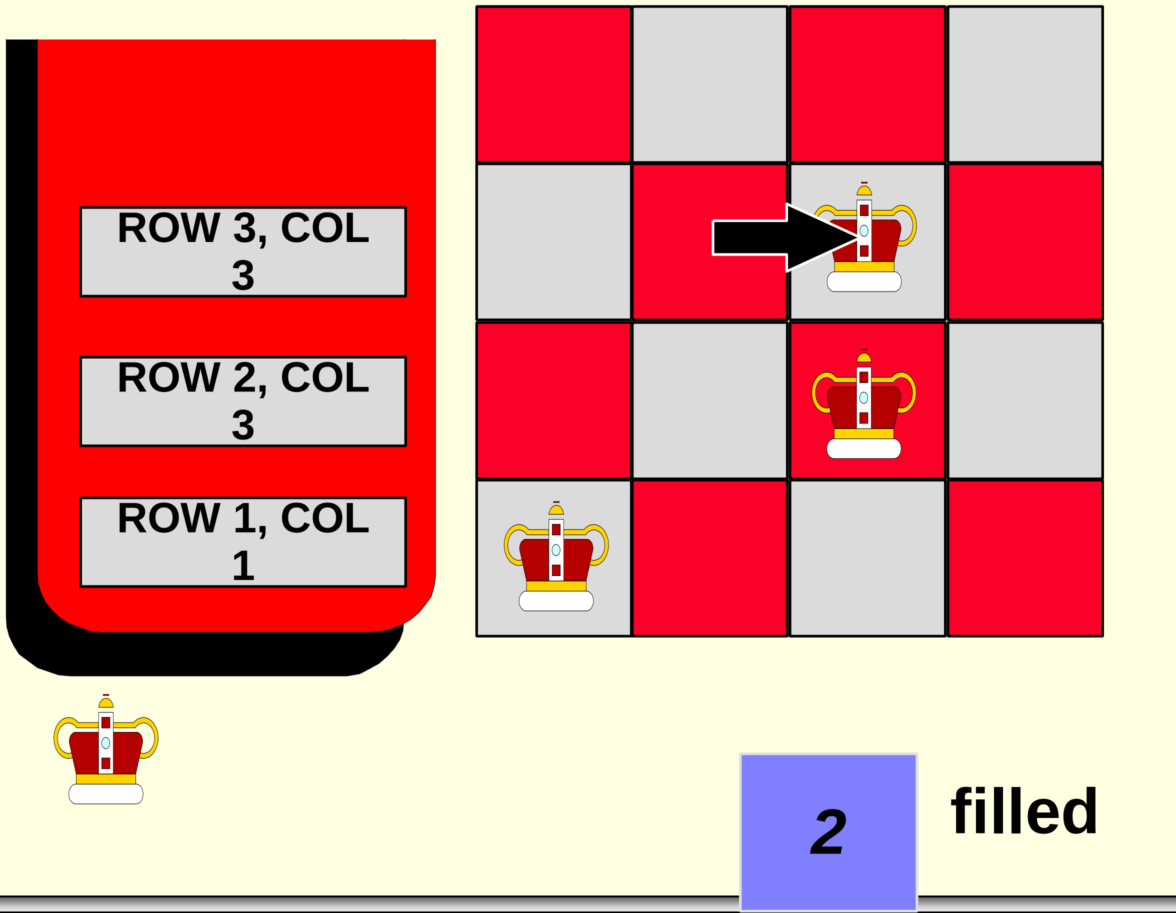


2 **filled**

How the program works

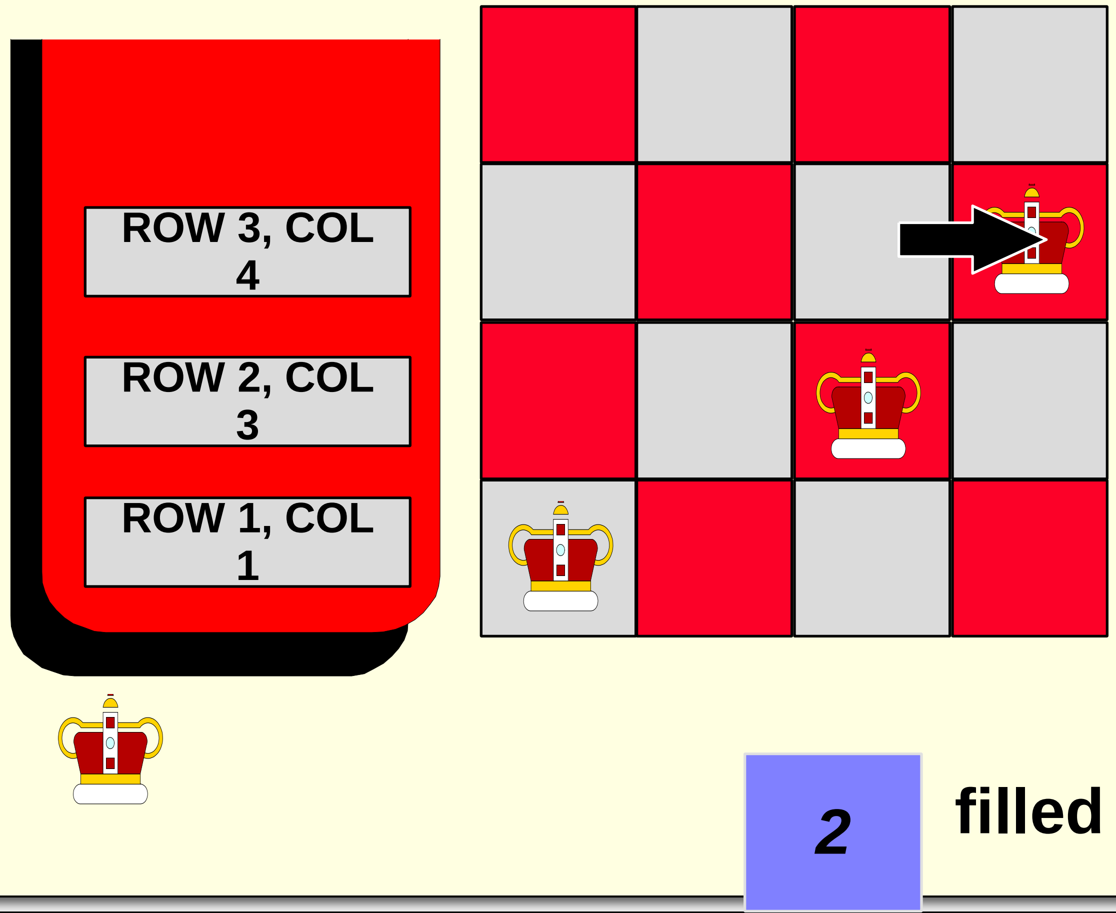
...and we shift to the third column.

Yet another conflict arises...



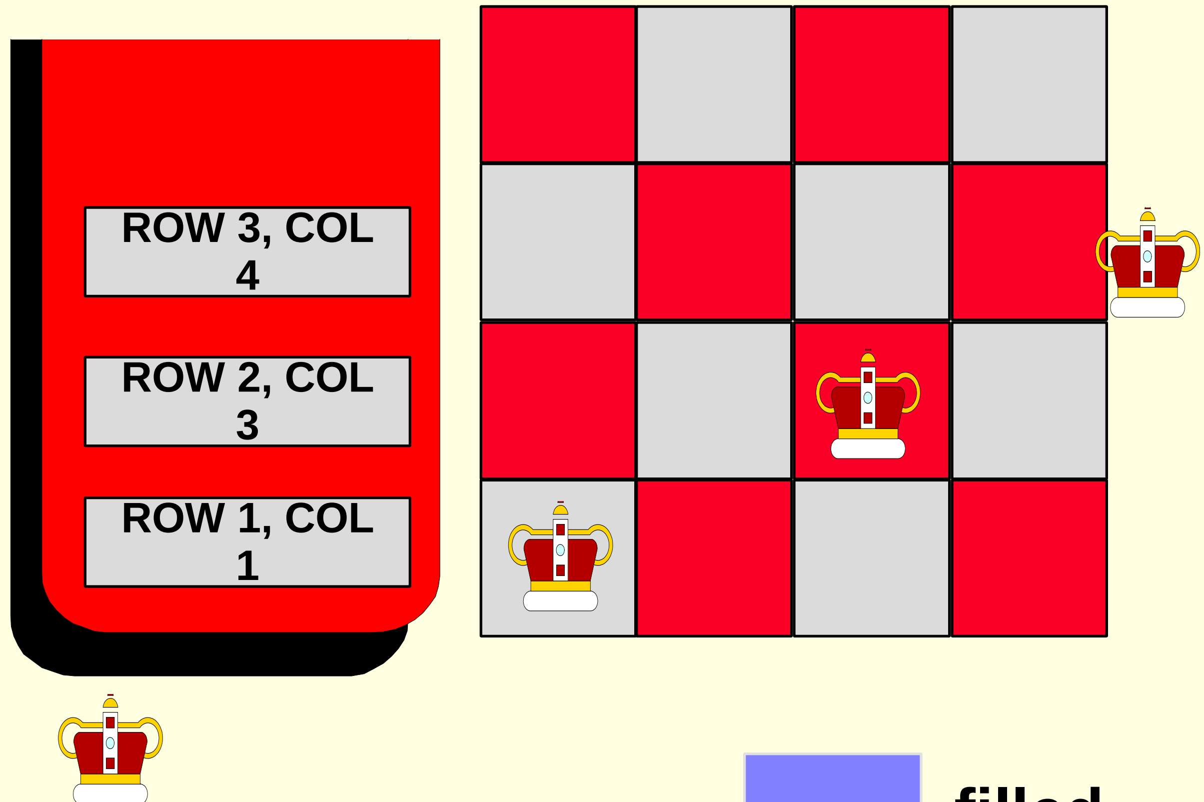
How the program works

...and we shift to column 4. There's still a conflict in column 4, so we try to shift rightward again...



How the program works

...but there's nowhere else to go.

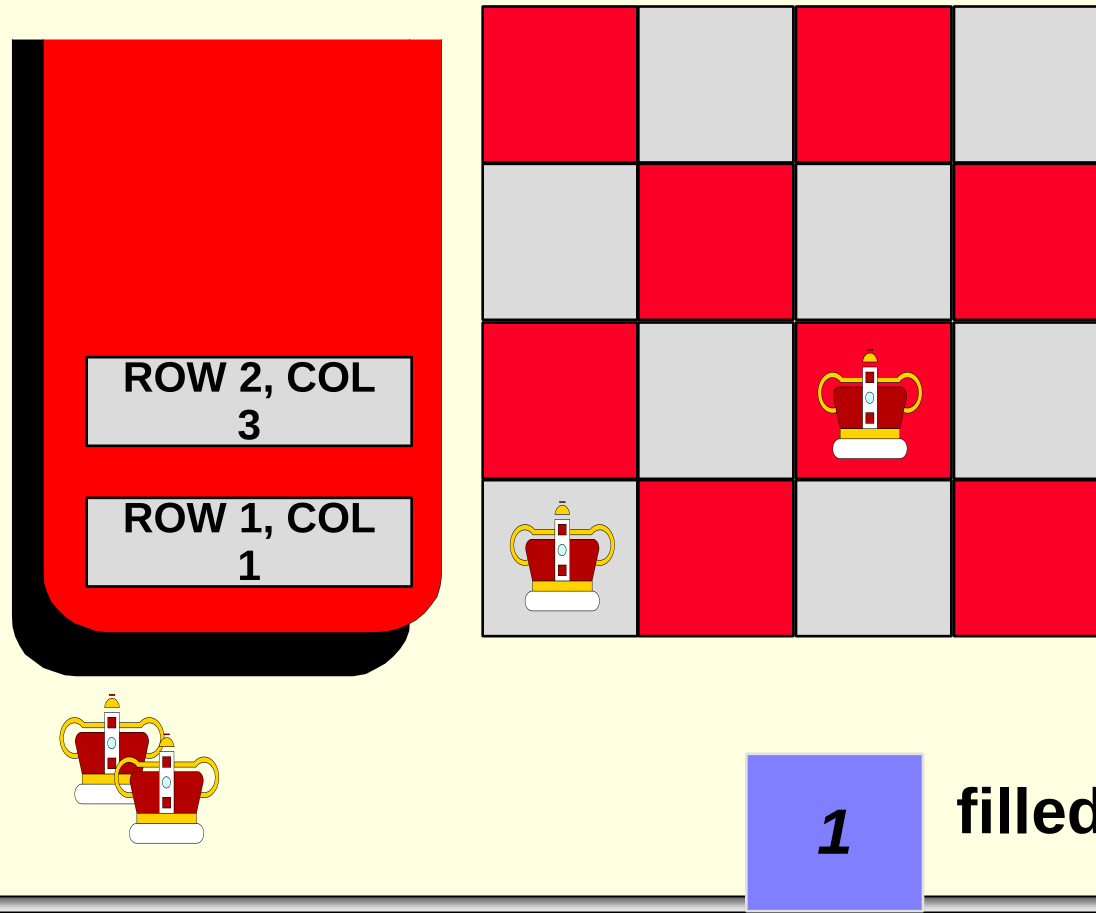


2 **filled**

How the program works

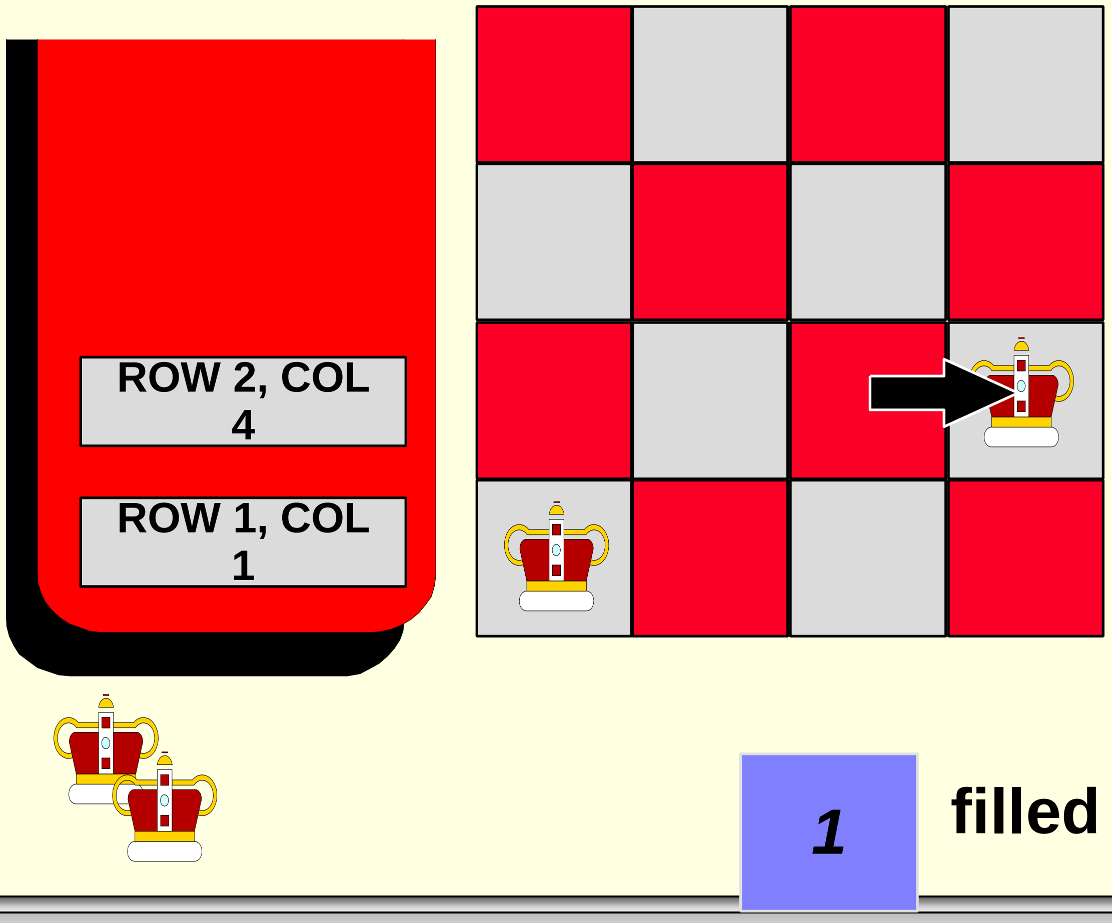
When we run out of room in a row:

- pop the stack,
- reduce filled by 1
- and continue working on the previous row.



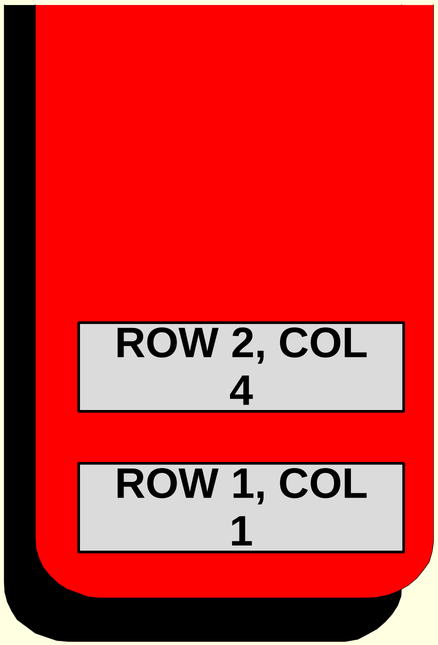
How the program works

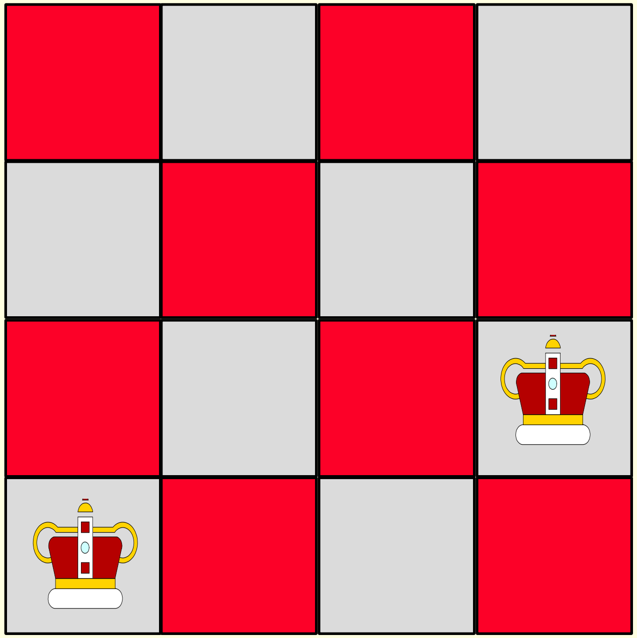
Now we continue working on row 2, shifting the queen to the right.

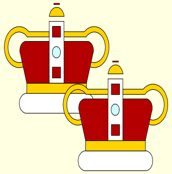


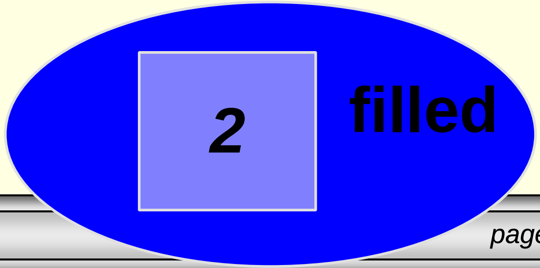
How the program works

This position has no conflicts, so we can increase filled by 1, and move to row 3.



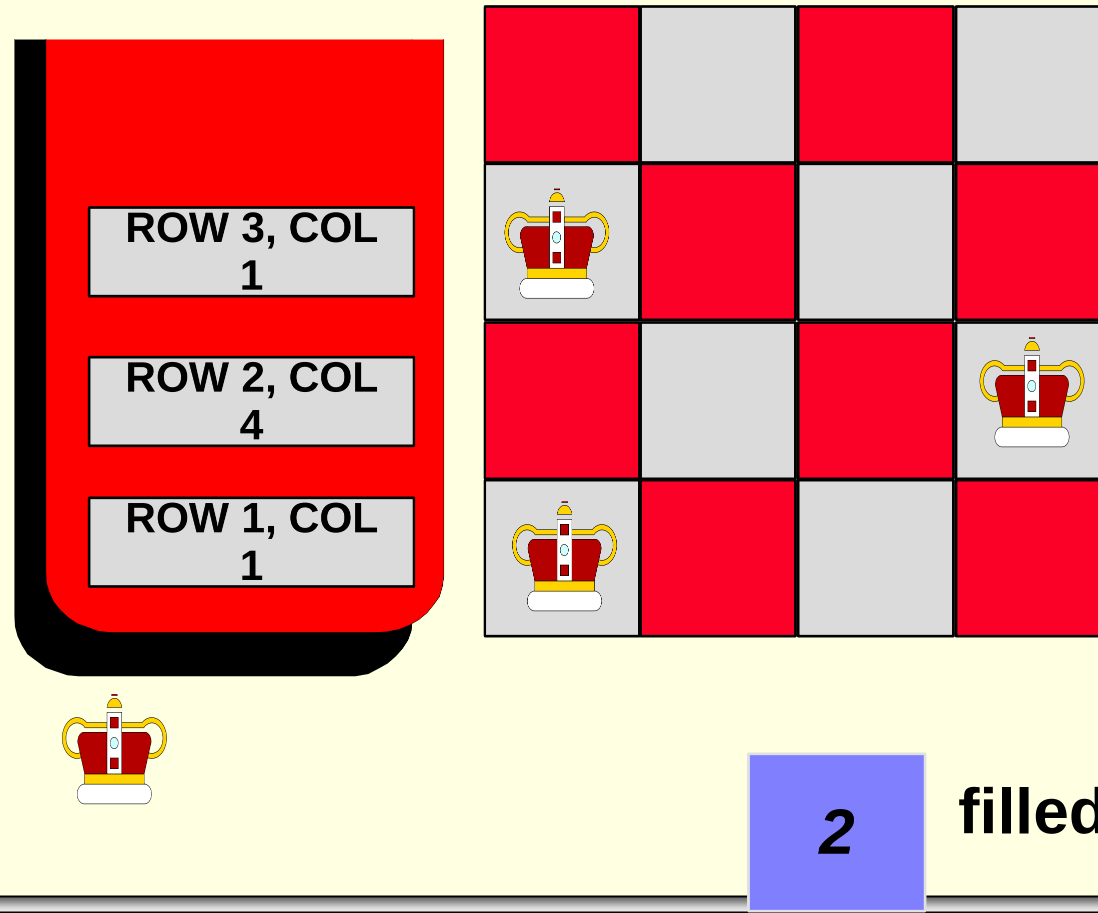






How the program works

In row 3, we start again at the first column.



Brief Interlude: FAIL Picture

Pseudocode for N-Queens

- Initialize a stack where we can keep track of our decisions.
- Place the first queen, pushing its position onto the stack and setting filled to 0.
- repeat these steps:
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...

Increase filled by 1. If filled is now N, then the algorithm is done. Otherwise, move to the next row and place a queen in the first column.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...

Move the current queen rightward, adjusting the record on top of the stack to indicate the new position.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

Backtrack!
Keep popping the stack, and reducing filled by 1, until you reach a row where the queen can be shifted rightward. Shift this queen right.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

Backtrack!
Keep popping the stack, and reducing filled by 1, until you reach a row where the queen can be shifted rightward. Shift this queen right.

Summary

- Stacks have many applications.
- The application which we have shown is called **backtracking**.
- The key to backtracking: Each choice is recorded in a stack.
- When you run out of choices for the current decision, you pop the stack, and continue trying different choices for the previous decision.
- Here's an applet to see nQueens in action:
  - http://www.cosc.canterbury.ac.nz/mukundan/dsal/NQP.html

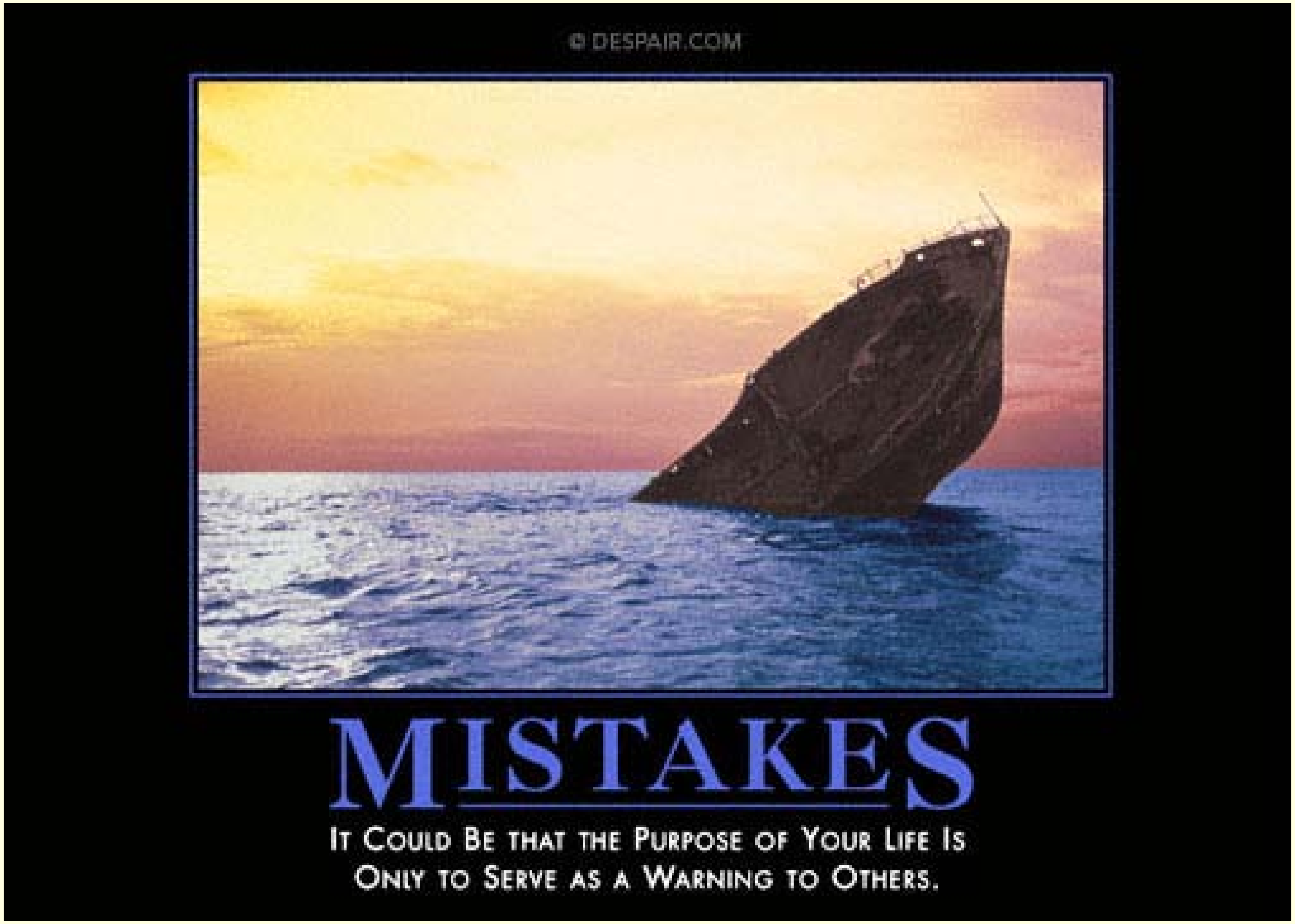
Backtracking

WASN'T

THAT

CAPTIVATING!

Daily Demotivator



Backtracking

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I