

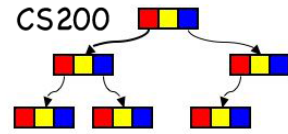
CS200:

# Recursion and induction (recap from cs161)

Prichard Ch. 6.1 & 6.3

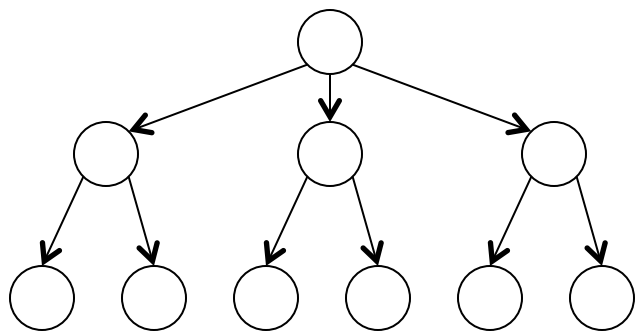


# Backtracking



- Problem solving technique that involves **moves: guesses** at a solution.
- **Depth First Search:** in case of failure retrace steps and try a new move in a state with still unexplored guesses

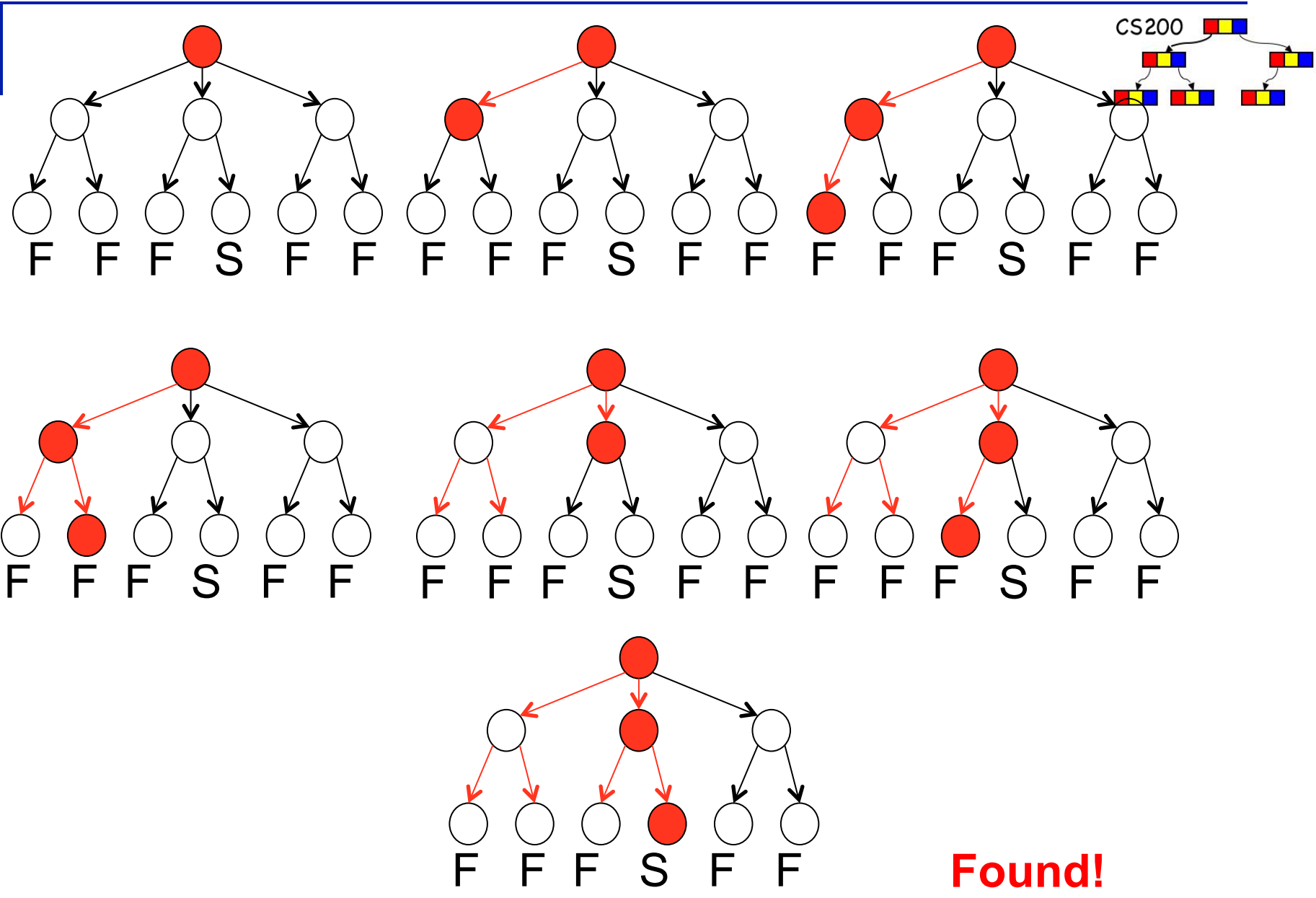
Think of it as walking through a tree shaped state space.



3 guesses here

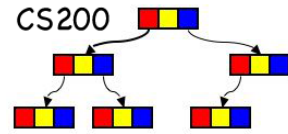
2 guesses in each state here

leaf states can fail (F) or succeed (S)

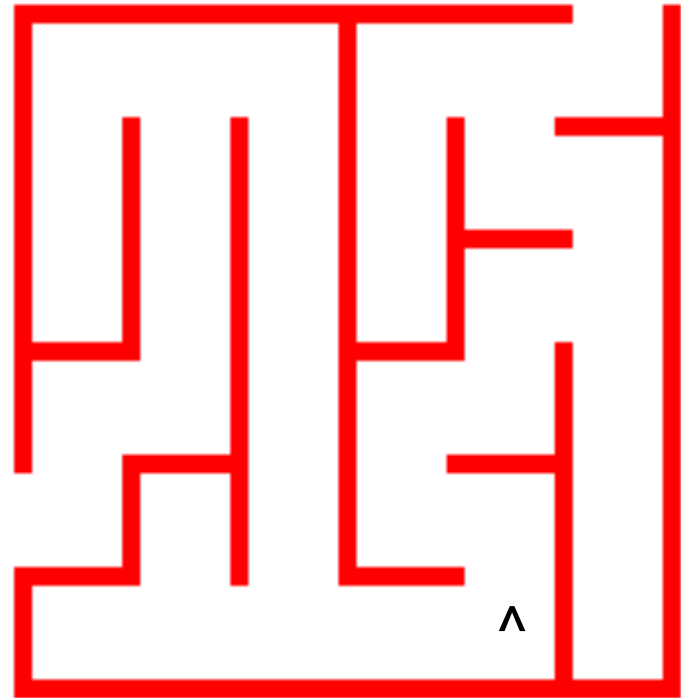


**Found!**

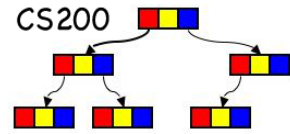
# Depth First Search



- Looking for a path out of the maze
- Strategy:
  - Prioritize directions: right, straight or left.
  - At a dead end “backtrack” and try a different direction
- Recursive solution?

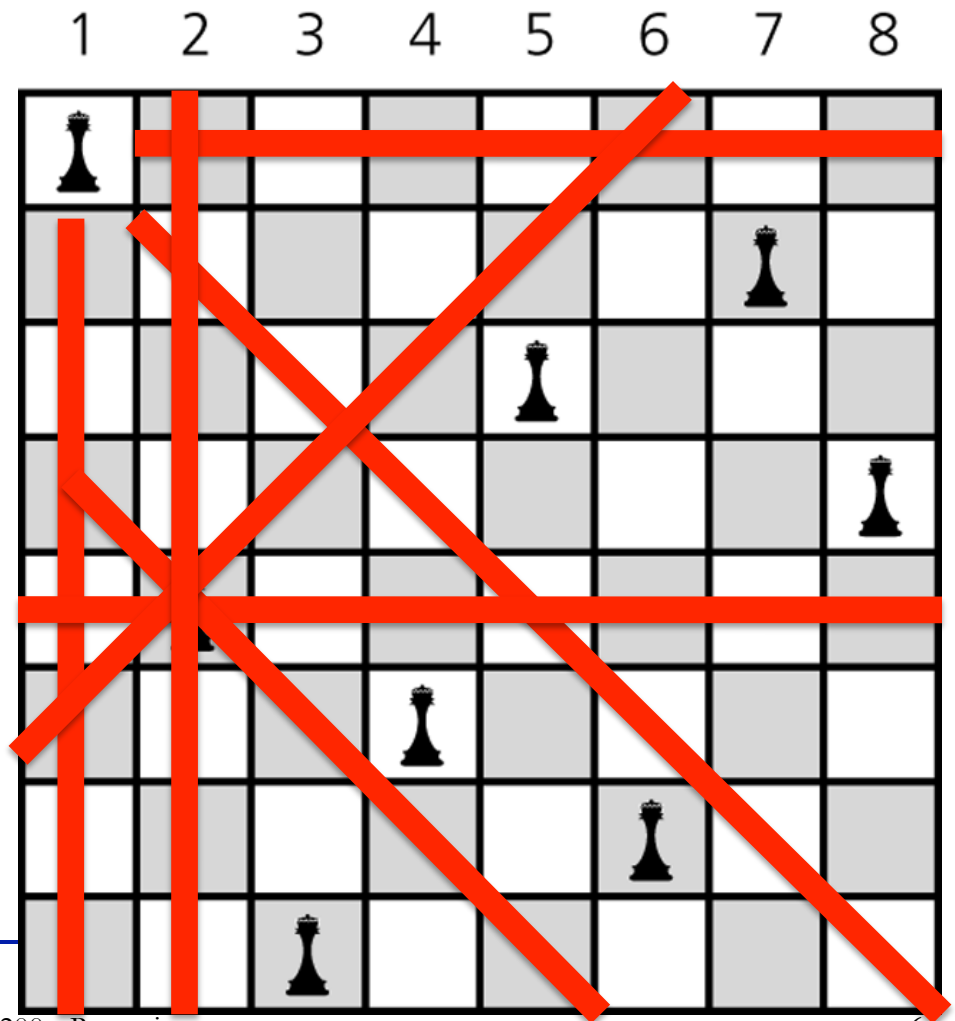


# The Eight Queens Problem

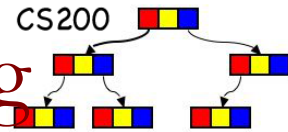


Place 8 Queens!

No queen can attack any other queens.

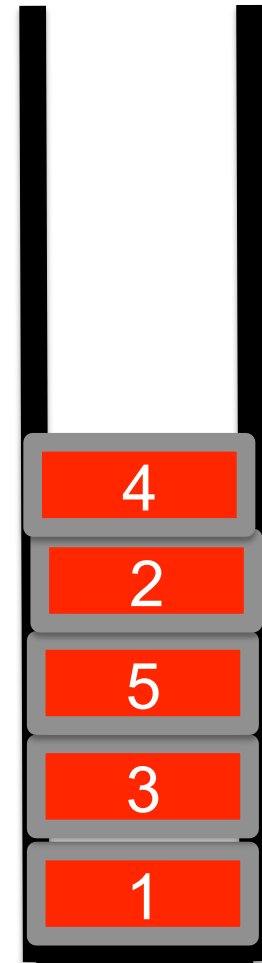
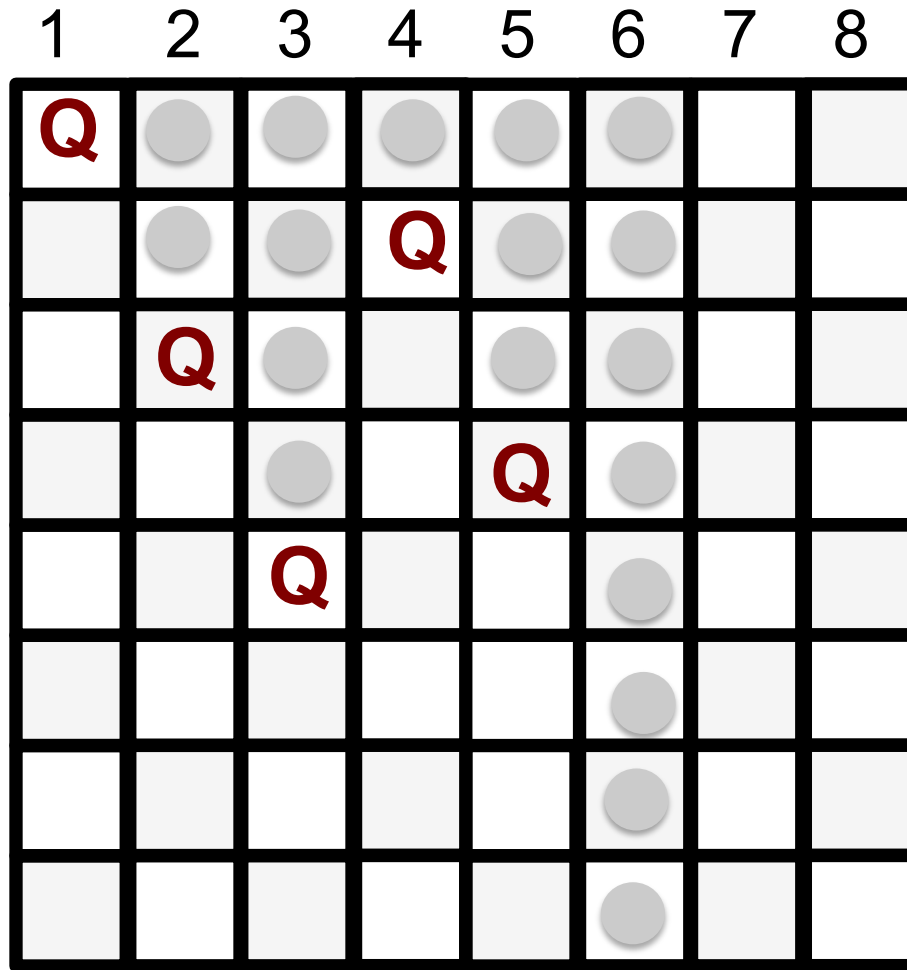
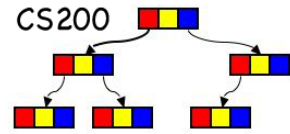


# Solution with recursion and backtracking



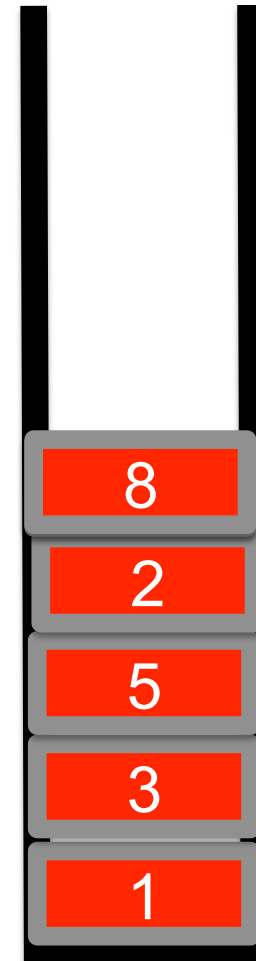
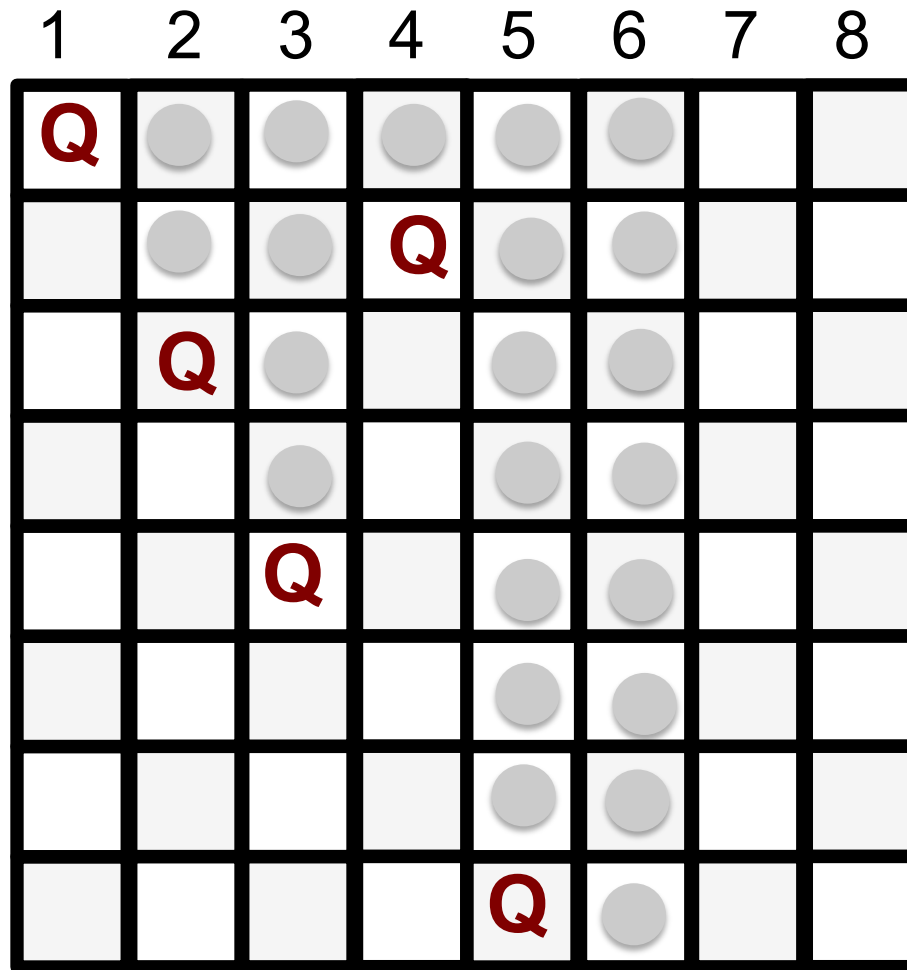
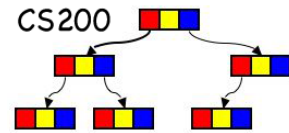
```
placeQueen (in currColumn:integer)
if ( currColumn > 8) {
    The problem is solved
} else {
    while (unconsidered squares exist in currColumn and the
        problem is unsolved) {
        Determine if the next square is safe.
        if (such a square exists){
            place a queen in the square
            placeQueens(currColumn+1) // try next column
            if (no queen safe in currColumn+1) {
                remove queen from currColumn
                try the next square in that column
            }
        }
    }
}
```

# Example



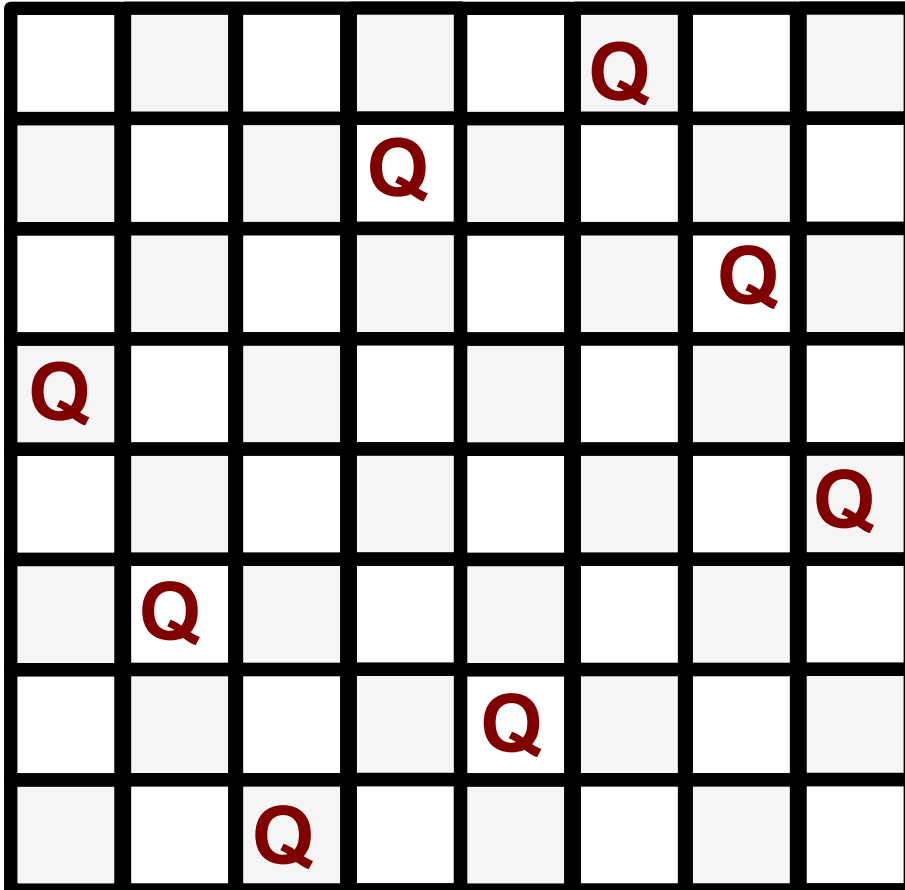
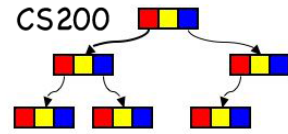


# Hit 'Dead End'





# Backtrack: an 8 queens solution



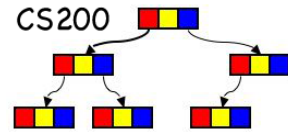
The only symmetric one

There are 11 more  
“fundamental” solutions

see:

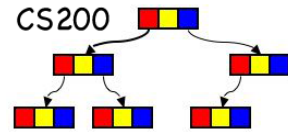
[wikipedia.org/wiki/  
Eight\\_queens\\_puzzle](http://wikipedia.org/wiki/Eight_queens_puzzle)

# Questions



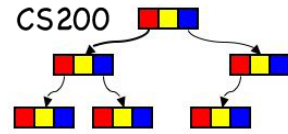
- What is the maximum depth of the run time stack for 8 Queens?
- How big could the call tree get?

# Recursion



- Specifies a solution to one or more base cases
- Then demonstrates how to derive the solution to a problem of an arbitrary size
  - From solutions to smaller sized problems.

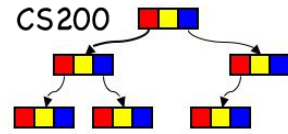
# Correctness of the Recursive Factorial Method



Specification of the problem  
(e.g., Mathematical definition, SW requirements)

Algorithm  
(e.g., pseudo code)

Does your algorithm satisfy the specification of the problem?



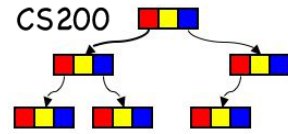
# Correctness of the Recursive Factorial Method

## Definition of Factorial

$factorial(n) = n (n - 1) (n - 2) \dots 1$  for any integer  $n > 0$   
 $factorial(0) = 1$

## Definition of method $fact(N)$

```
1: fact (in n: integer): integer
2:     if (n is 0) {
3:         return 1
4:     } else {
5:         return n* fact(n-1)
6:     }
```



Inductive proof fact computes the factorial of its argument.

**Basis step:**

$$\text{fact}(0) = 1$$

**Inductive Step:**

Show that for an arbitrary positive integer  $k$ ,

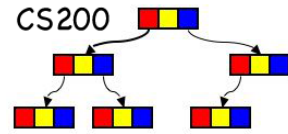
if  $\text{fact}(k)$  returns  $k!$ , then

$\text{fact}(k+1)$  returns  $(k+1)!$

*do it do it*



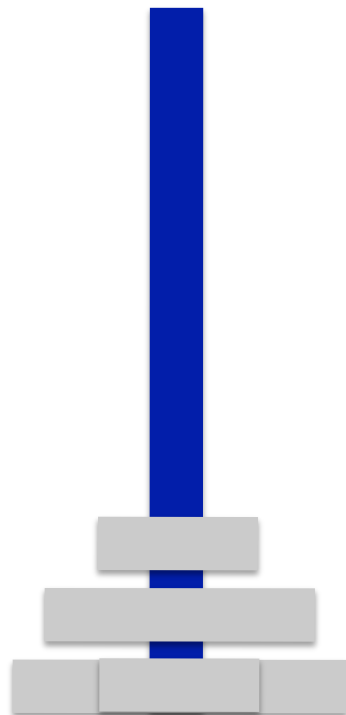
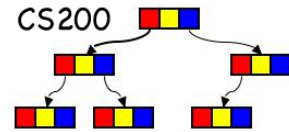
# The Towers of Hanoi Example



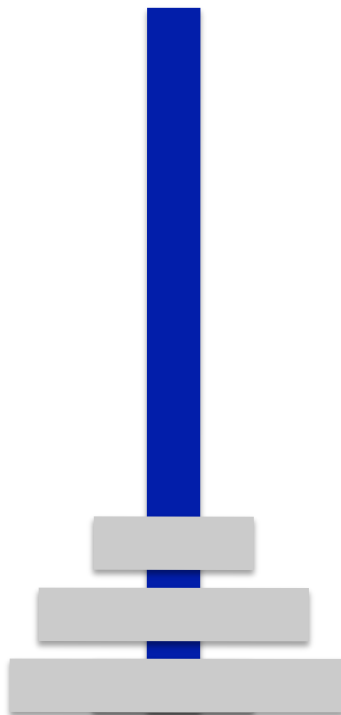
- **Move pile of disks from source to destination**
- **Only one** disk may be moved at a time.
- No disk may be placed on top of a smaller disk.



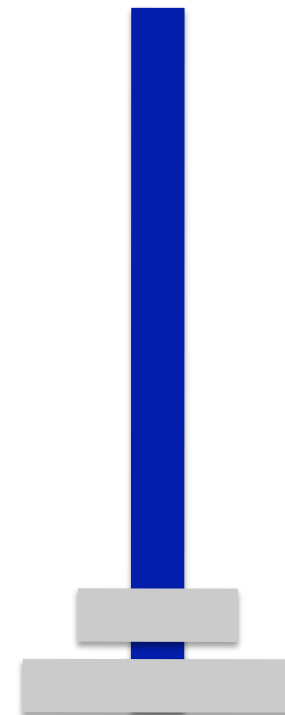
# States in the Towers of Hanoi



Source

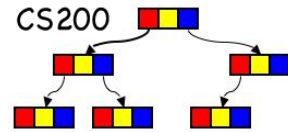


Destination



Spare

# Recursive Solution



```
// pegs are numbers, via is computed
```

```
// number of moves are counted
```

```
// empty base case
```

```
public void hanoi(int n, int from, int to){
```

```
    if (n>0) {
```

```
        int via = 6 - from - to;
```

```
        hanoi(n-1,from, via);
```

```
        System.out.println("move disk " + n + " from " + from + " to " + to);
```

```
        hanoi(n-1,via,to);
```

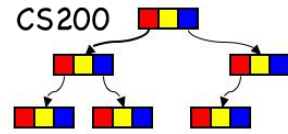
```
    }
```

```
}
```

let's run it and

study the move pattern, and count the number of moves

# Cost of Towers of Hanoi



- How many moves does  $hanoi(n)$  make?
- *from the recursive code:*

$$moves(1) = 1$$

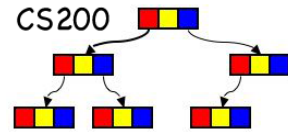
$$moves(N) = moves(N-1) + 1 + moves(N-1) \text{ (if } N > 1)$$

- By inspection, we can infer that a closed form formula for the number of moves:

$$moves(N) = 2^N - 1 \text{ (for all } N \geq 1)$$

- **Can we prove it?**

# Proof



## ■ Basis Step

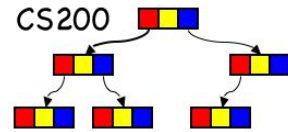
- Show that the property is true for  $N = 1$ .

$2^1 - 1 = 1$ , which is consistent with the recurrence relation's specification that  $moves(1) = 1$

## ■ Inductive Step

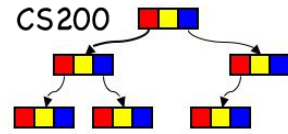
- Property is true for an arbitrary  $k \rightarrow$  property is true for  $k+1$
- Assume that the property is true for  $N = k$   
 $moves(k) = 2^k - 1$
- Show that the property is true for  $N = k + 1$
- *Do it, do it*

# Proof – cont.



- $moves(k+1) = 2 * moves(k) + 1$   
 $= 2 * (2^k - 1) + 1$   
 $= 2 * 2^k - 2 + 1 = 2^{k+1} - 1$

Therefore the inductive proof is complete.



One more example:

$$0+1+2\dots+n = n(n+1)/2 \quad n=0,1,2,\dots$$

**base:**  $0 = 0*1/2=0$  Check

**step:** assume:  $0+1+2\dots+k = k(k+1)/2$

show that  $0+1+2\dots+k+ (k+1) = (k+1)(k+2)/2$

$$0+1+2\dots+k+ (k+1) = k(k+1)/2 + (k+1) =$$

$$k(k+1)/2 + 2(k+1)/2 = k(k+1)/2 + 2(k+1)/2 =$$

$$(k+2)(k+1)/2 = (k+1)(k+2)/2$$

Check