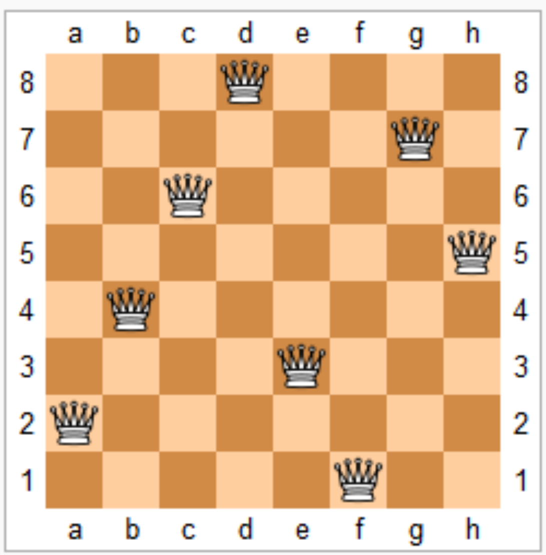
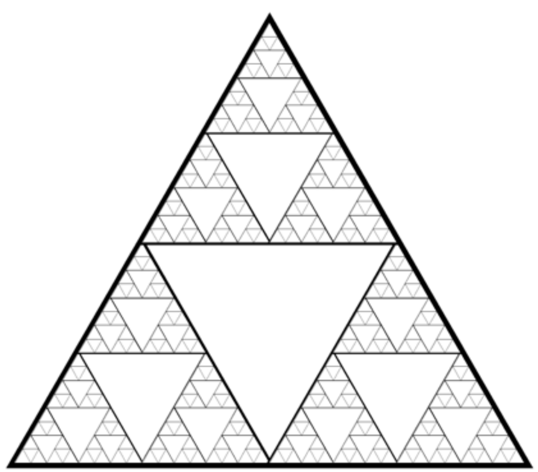
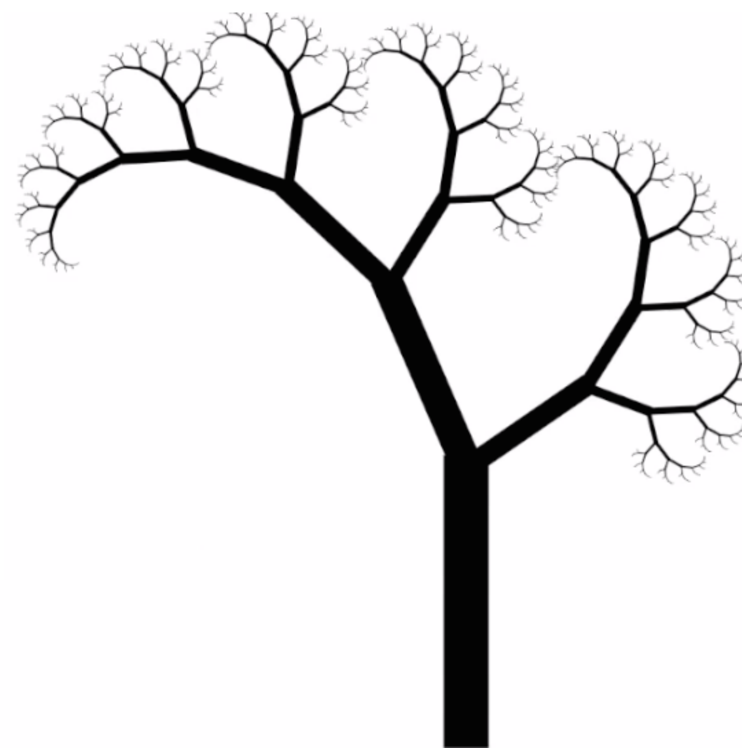
15-112
Fundamentals of Programming

Week 5 - Lecture 3:
More Advanced Recursion



One solution to the eight queens puzzle





June 15, 2016

Recursion vs Iteration

15-112 View

	Recursion	Iteration
Elegance	+	-
Performance	-	+
Debugability	-	+

Memoization

```
def fib(n):
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    return result

print(fib(6))
```

How many times is fib(2) computed? **5**

Memoization

```
fibResults = dict()

def fib(n):
    if (n in fibResults):
        return fibResults[n]
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    fibResults[n] = result
    return result
```

Expanding the stack size and recursion limit

```
def rangeSum(lo, hi):
    if (lo > hi):
        return 0
    else:
        return lo + rangeSum(lo+1, hi)

print(rangeSum(1, 1234))
# RuntimeError: maximum recursion depth exceeded



print(callWithLargeStack(rangeSum(1, 123456)))
 # Works
```

More Examples

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

All subsets = All subsets that do not contain 1 +

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

All subsets = All subsets that do not contain 1 +

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

All subsets = All subsets that do not contain 1 +

All subsets that contain 1

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

[1] + subset that doesn't contain a 1

All subsets = All subsets that do not contain 1 +

All subsets that contain 1

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

```
def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = [ ]
        for subset in powerset(a[1:]):
            allSubsets += [subset]
            allSubsets += [[a[0]] + subset]
        return allSubsets
```

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

```
def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = [ ]
        for subset in powerset(a[1:]):
            allSubsets += [subset]
            allSubsets += [[a[0]] + subset]
        return allSubsets
```

Power set

Given a list, return a list of all the subsets of the list.

[1,2,3] -> [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

```
def powerset(a):
  if (len(a) == 0):
      return [[]]
  else:
      allSubsets = [ ]
      for subset in powerset(a[1:]):
          allSubsets += [subset]
          allSubsets += [[a[0]] + subset]
      return allSubsets
```

Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

[1,2,3], [2,1,3], [2,3,1]

Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

[1,2,3], [2,1,3], [2,3,1] [1,3,2], [3,1,2], [3,2,1]

Permutations

Permutations of [2,3,4]

[2,3,4]

[1,2,3,4]
[2,1,3,4]
[2,3,1,4]
[2,3,4,1]

[2,4,3]

[1,2,4,3]
[2,1,4,3]
[2,4,1,3]
[2,4,3,1]

[1,2,3,4]

[3,2,4]

[3,4,2]

[4,2,3]

[4,3,2]

• •

• •

• •

Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

```
def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = [ ]
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
```

Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

```
def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = [ ]
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
```

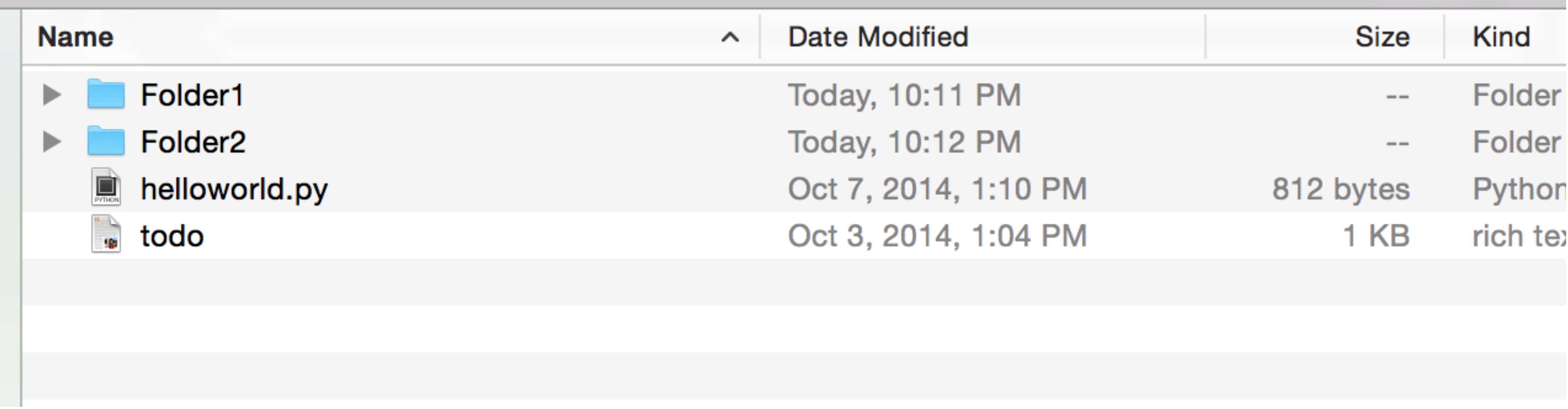
Permutations

Given a list, return all permutations of the list.

[1,2,3] -> [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

```
def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = [ ]
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
```

Print files in a directory



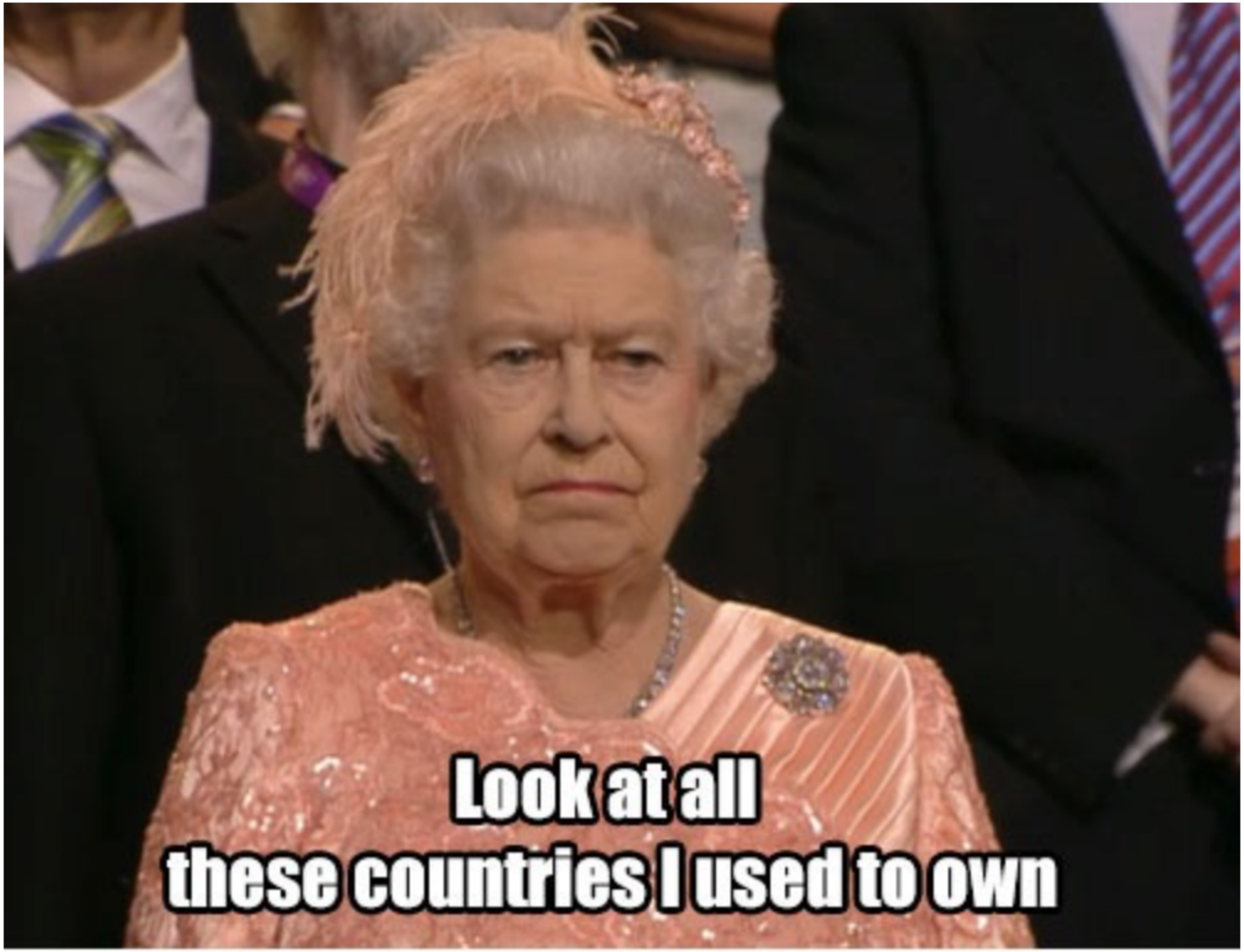
Print files in a directory

Name	Date Modified	Size	Kind
▼ Folder1	Today, 10:11 PM	--	Folder
foo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
fooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
▼ SubFolder1	Today, 10:11 PM	--	Folder
foooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
▼ SubFolder2	Today, 10:12 PM	--	Folder
fooooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
foooooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
▼ SubSubFolder1	Today, 10:13 PM	--	Folder
somePic	Today, 9:32 PM	56 KB	PNG i
▼ Folder2	Today, 10:12 PM	--	Folder
haha	Oct 3, 2014, 1:04 PM	1 KB	rich te
helloworld.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
todo	Oct 3, 2014, 1:04 PM	1 KB	rich te

Print files in a directory

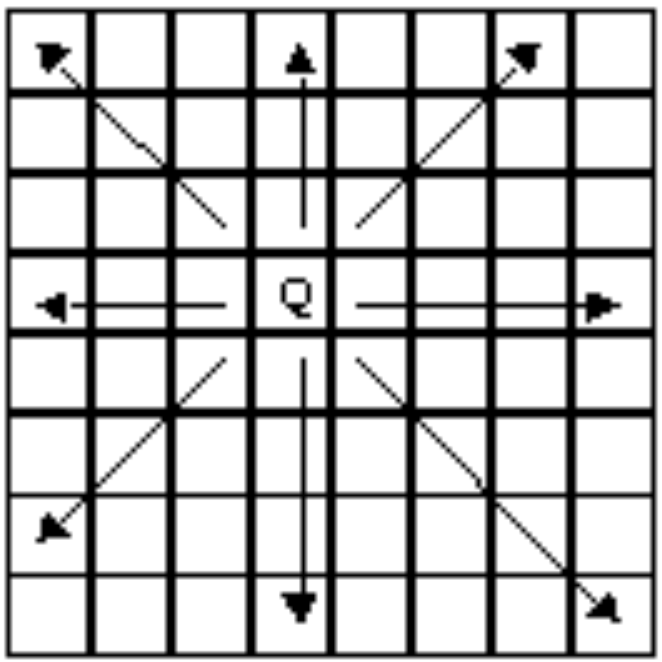
```
import os
def printFiles(path):
    if (os.path.isdir(path) == False):
        # base case:  not a folder, but a file, so print its path
        print(path)
    else:
        # recursive case: it's a folder
        for filename in os.listdir(path):
            printFiles(path + "/" + filename)
```

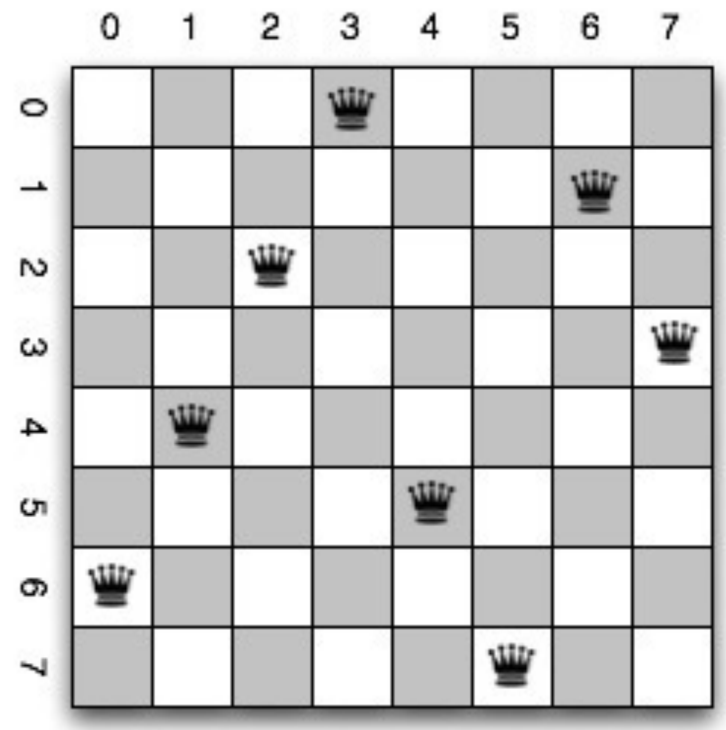
nQueens Problem



nQueens Problem

Place n queens on a n by n board so that no queen is attacking another queen.





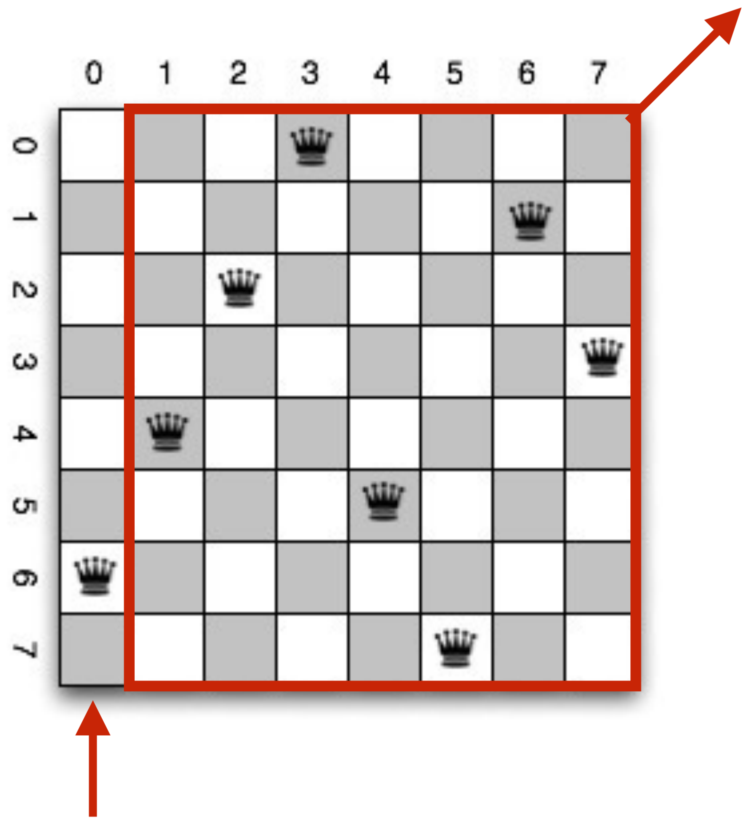
def solve(n): —> [6, 4, 2, 0, 5, 7, 1, 3]

list of rows

nQueens Problem

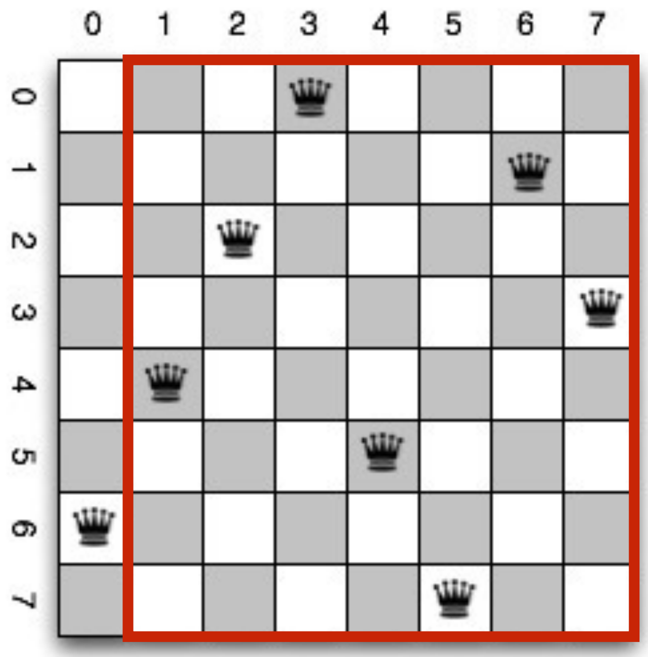
Place n queens on a n by n board so that no queen is attacking another queen.

n rows and n-1 columns



one queen has to be on first column

nQueens Problem



__First attempt:__

- try rows 0 to 7 for first queen
- for each try, recursively solve the red part

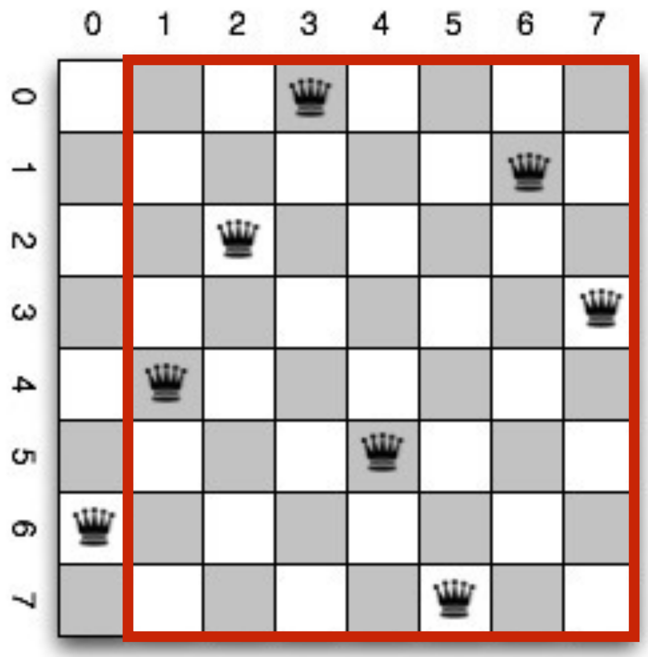
__Problem:__

Can't solve red part without taking into account first queen
First queen puts constraints on the solution to the red part

Need to be able to solve nQueens with added constraints.
Need to generalize our function:

```
def solve(n, m, constraints):
```

nQueens Problem



def solve(n, m, constraints):

n = number or rows

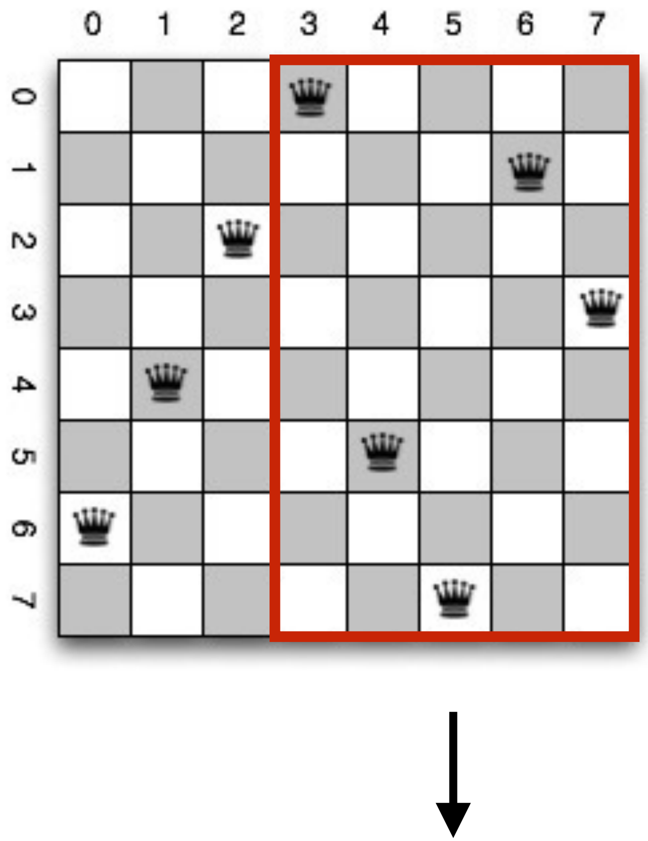
m = number or columns

constraints (in what form?)

list of rows

For the red part, we have the constraint [6]

nQueens Problem



def solve(n, m, constraints):

n = number or rows

m = number or columns

constraints (in what form?)

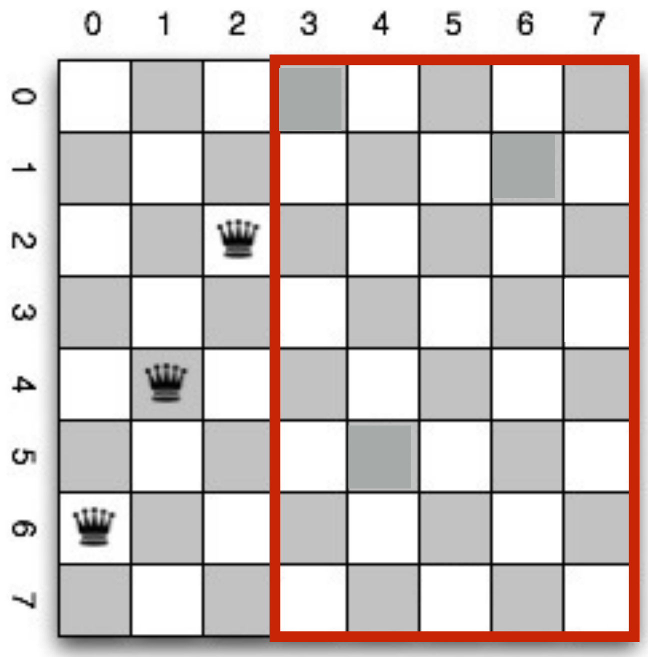
list of rows

For the red part, we have the constraint [6,4,2]

The constraint tells us which cells are <u>unusable</u> for the red part.

To solve original nQueens problem, call: solve(n, n, [])

nQueens Problem



[?,?,?,?,?]

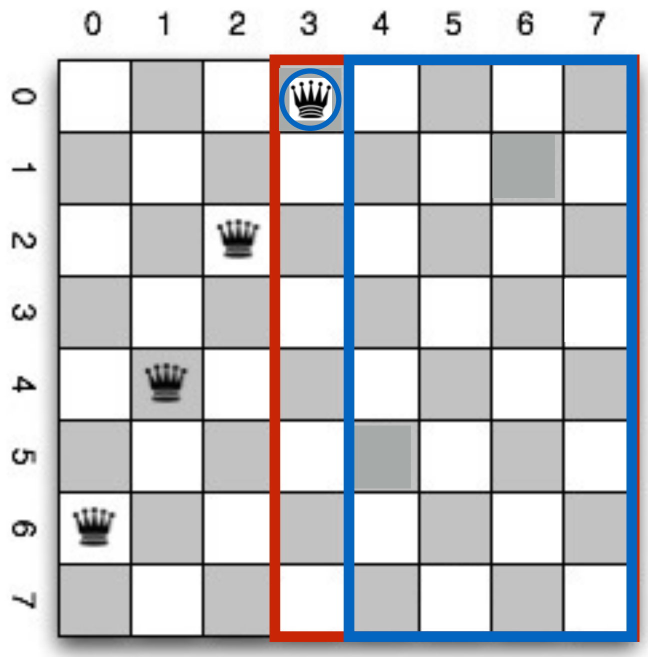
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

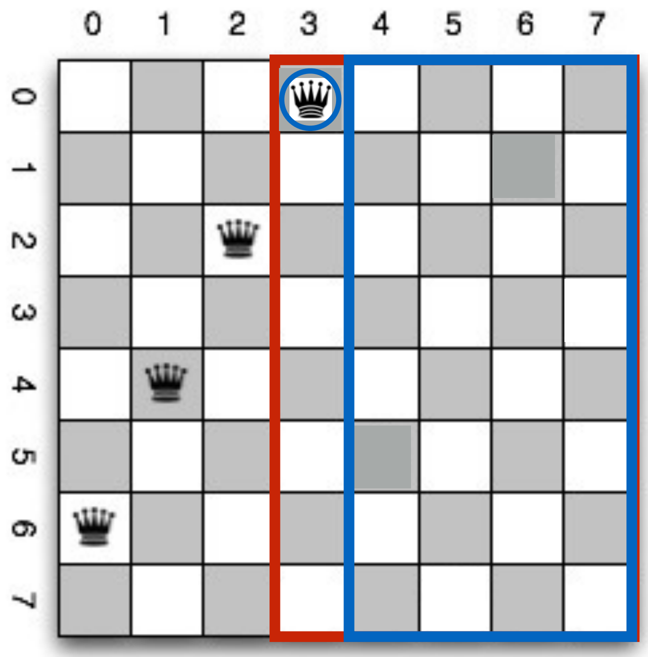
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

[5,7,1,3]

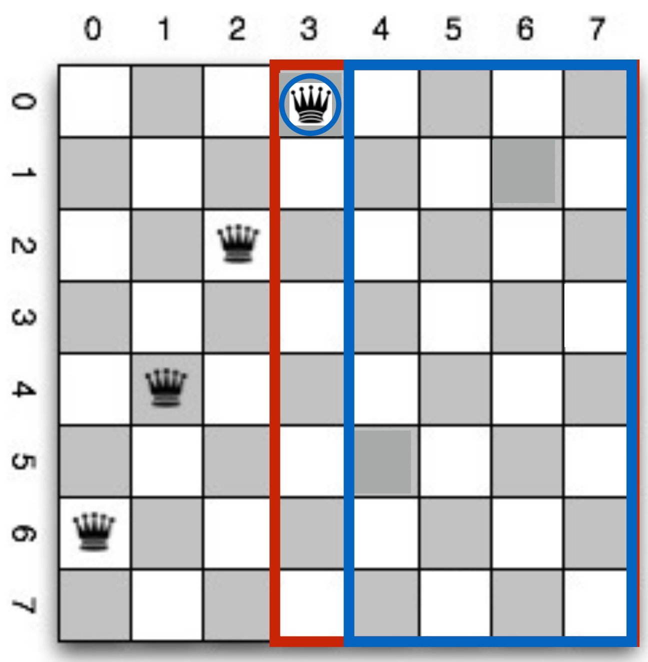
n = 8

m = 5

constraints = [6,4,2]

```
def solve(n, m, constraints):
```

nQueens Problem



[0,?,?,?,?]

[5,7,1,3] —> [0,5,7,1,3]

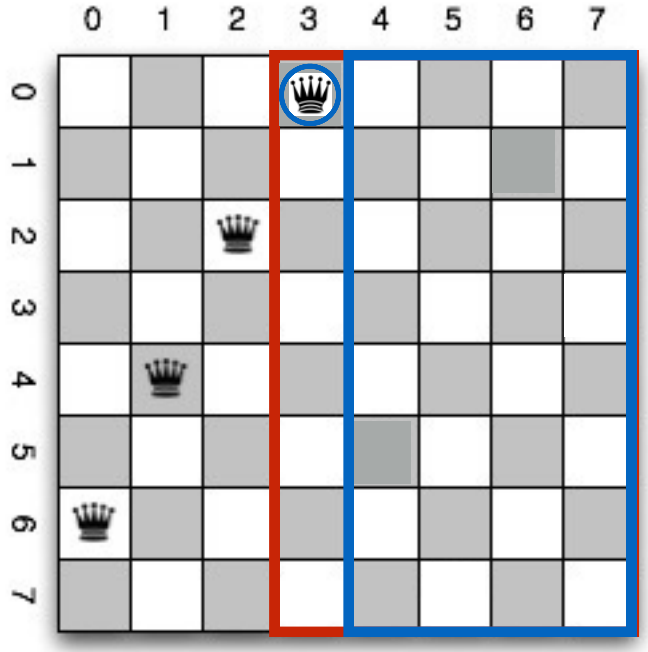
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

Suppose no solution

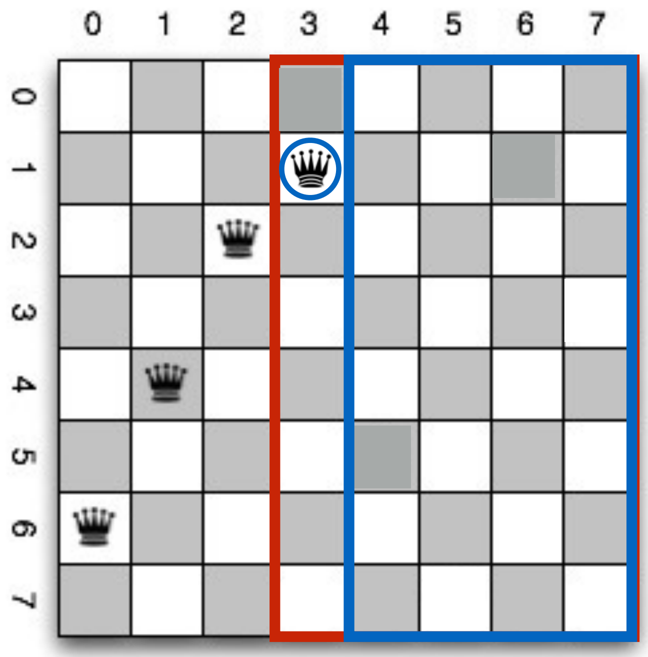
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

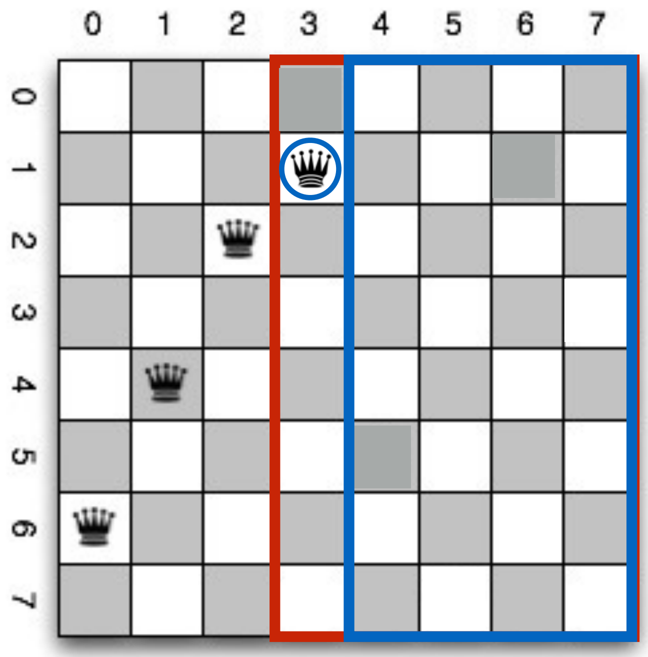
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

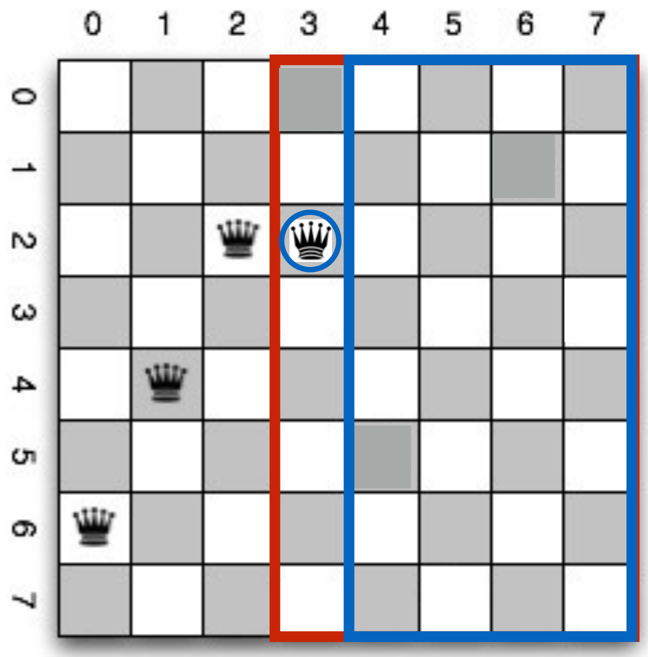
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

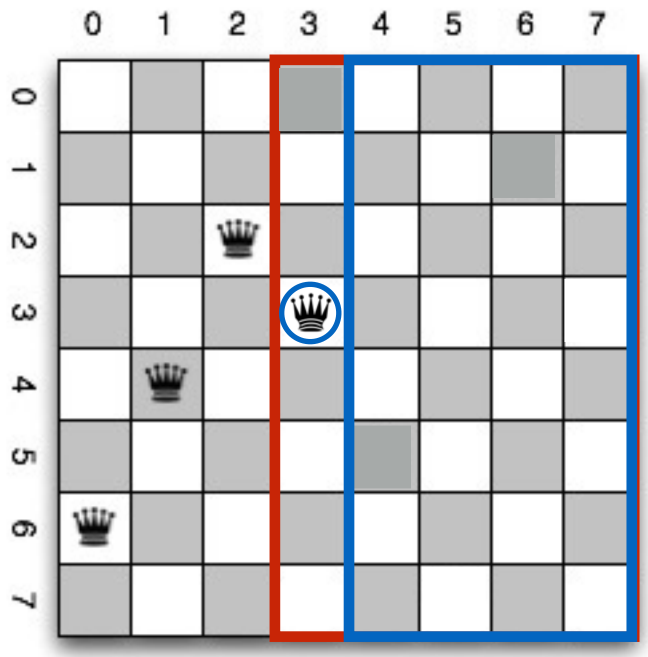
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

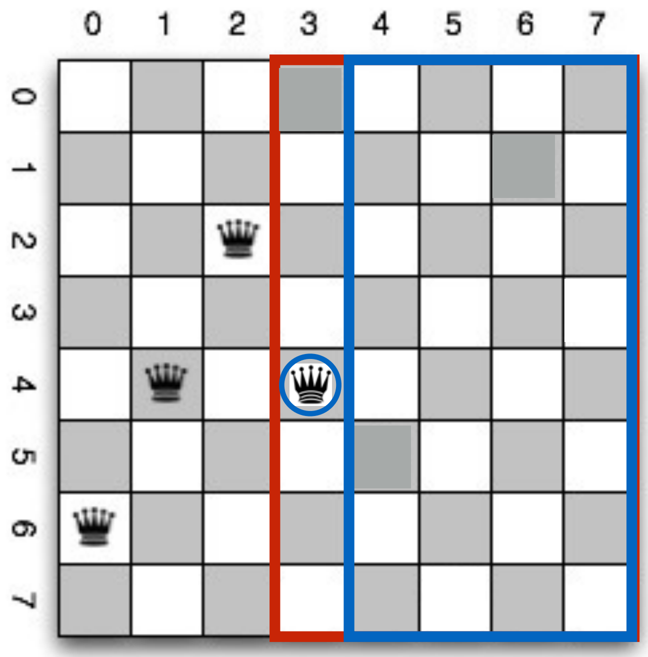
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

```
def solve(n, m, constraints):
```

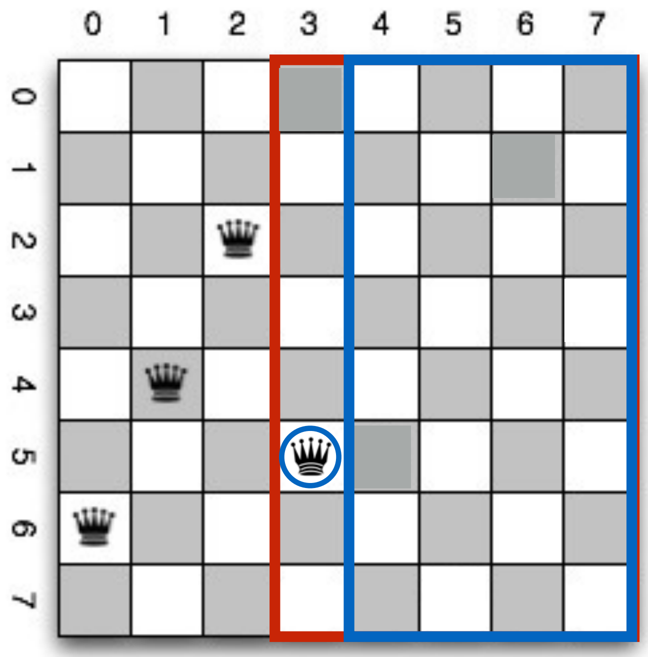
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

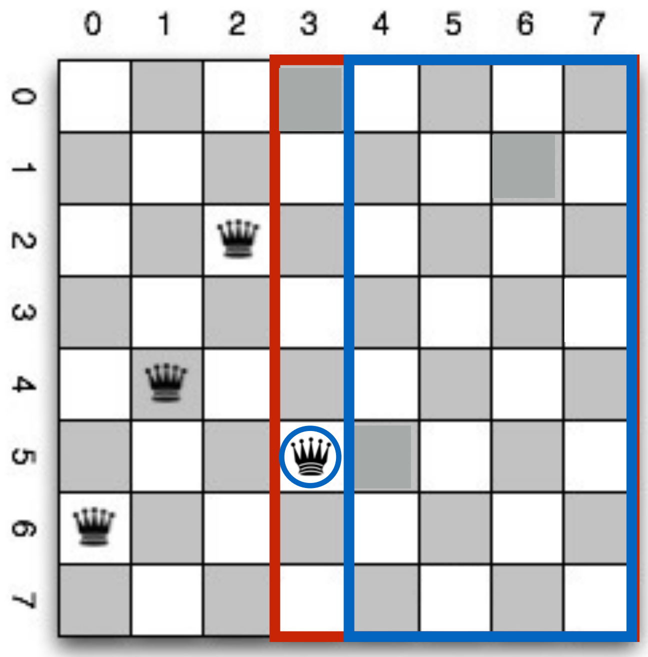
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

no solution

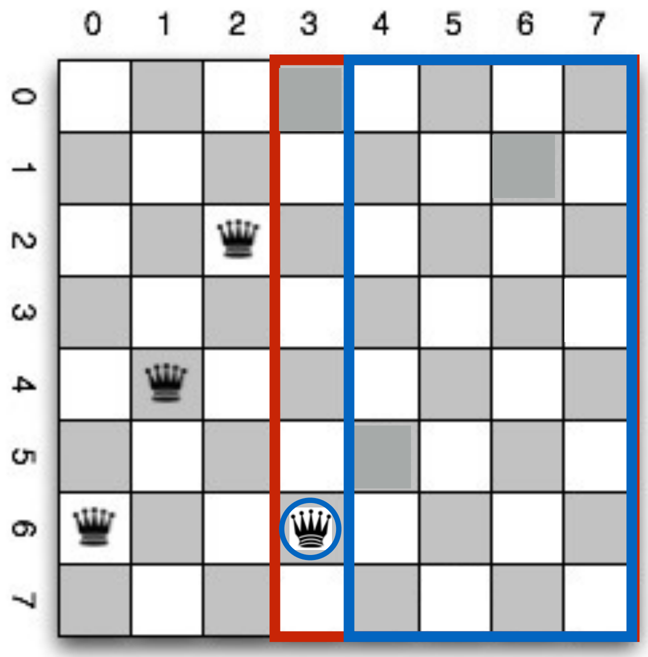
def solve(n, m, constraints):

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

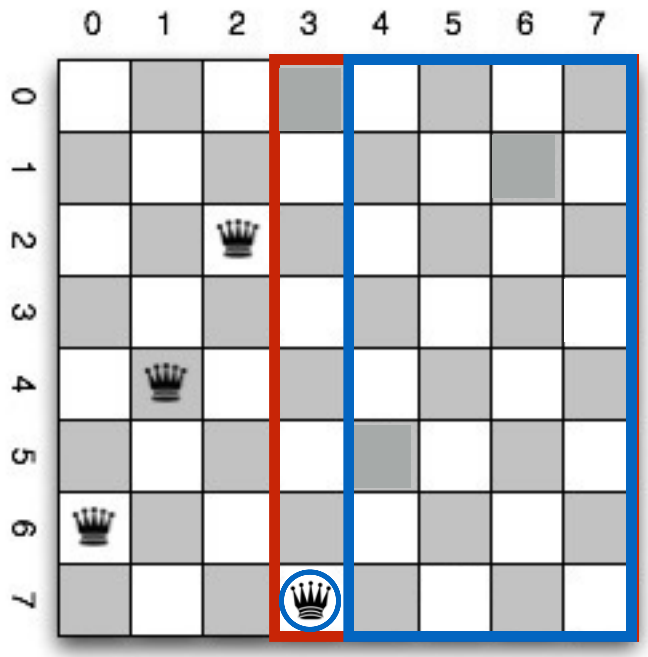
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

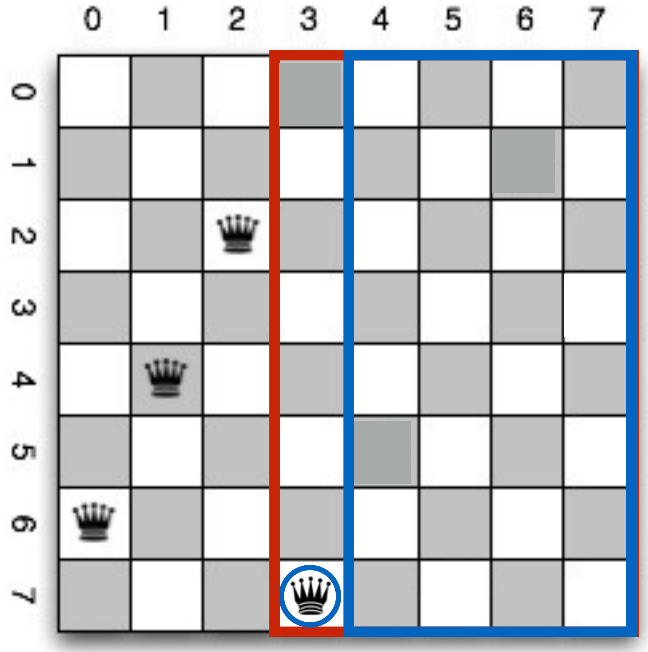
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

no solution

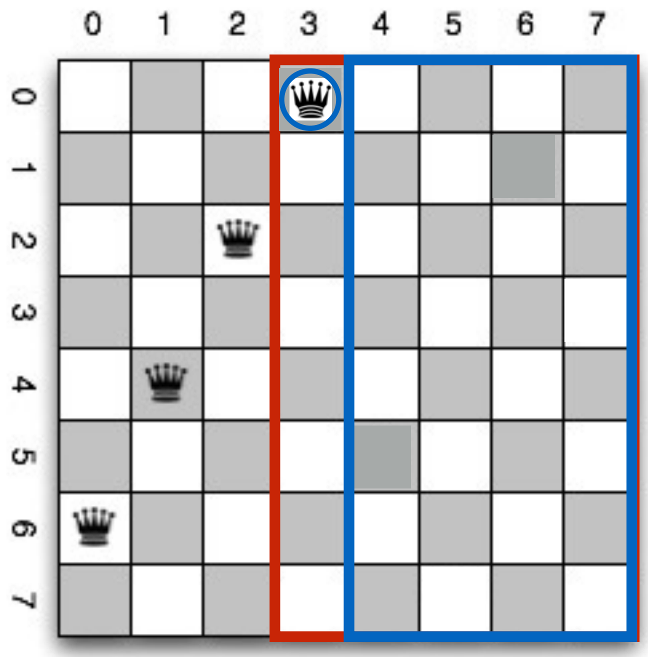
n = 8

m = 5

constraints = [6,4,2]

```
def solve(n, m, constraints):
```

nQueens Problem



[0,?,?,?,?]

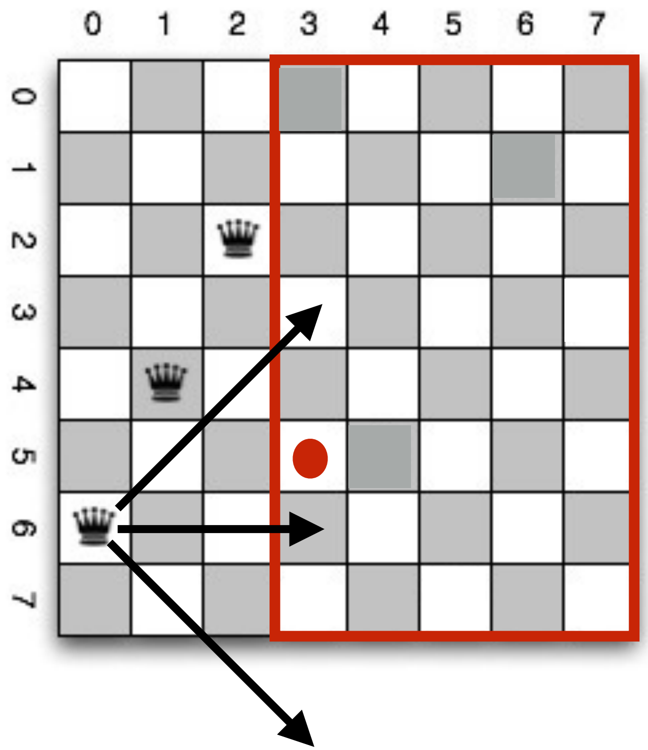
```
def solve(n, m, constraints):
  if(m == 0):
    return []
  for row in range(n):
    if (isLegal(row, constraints)):
      newConstraints = constraints + [row]
      result = solve(n, m-1, newConstraints)
      if (result != False):
        return [row] + result
  return False
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



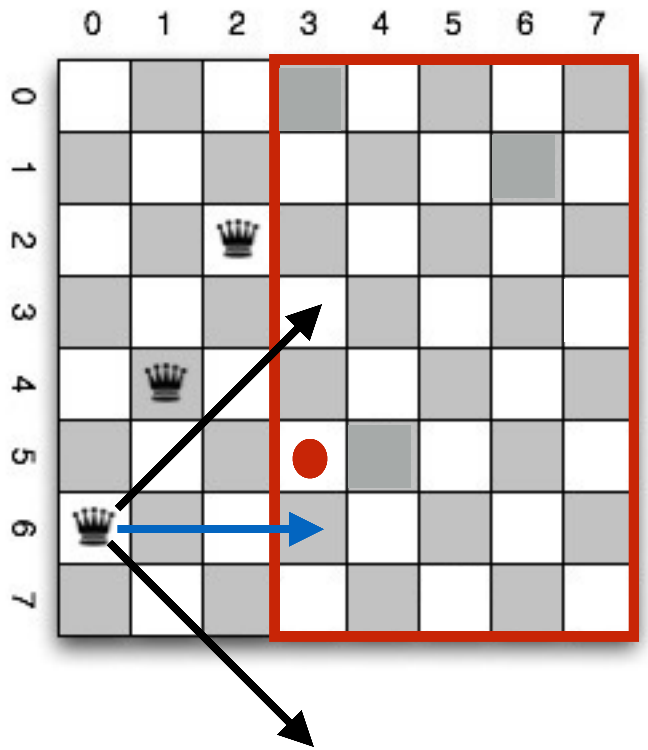

```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



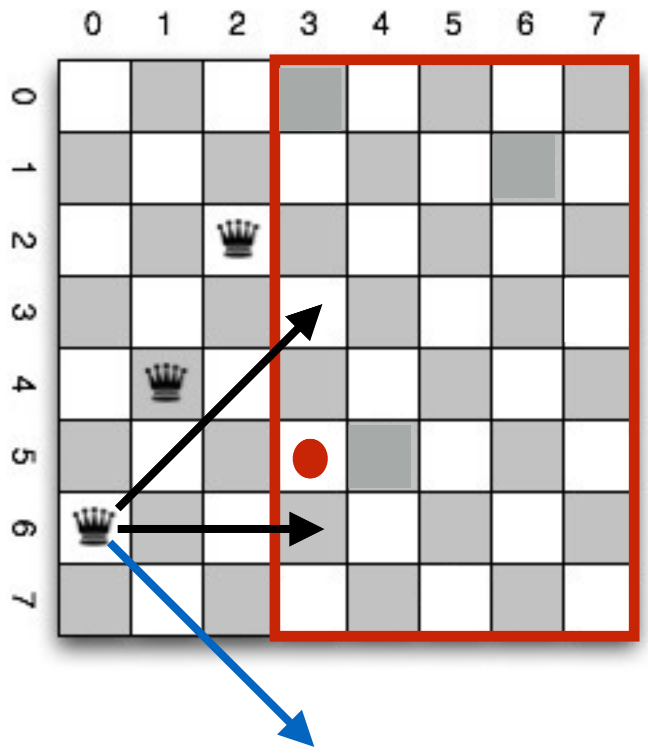

```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



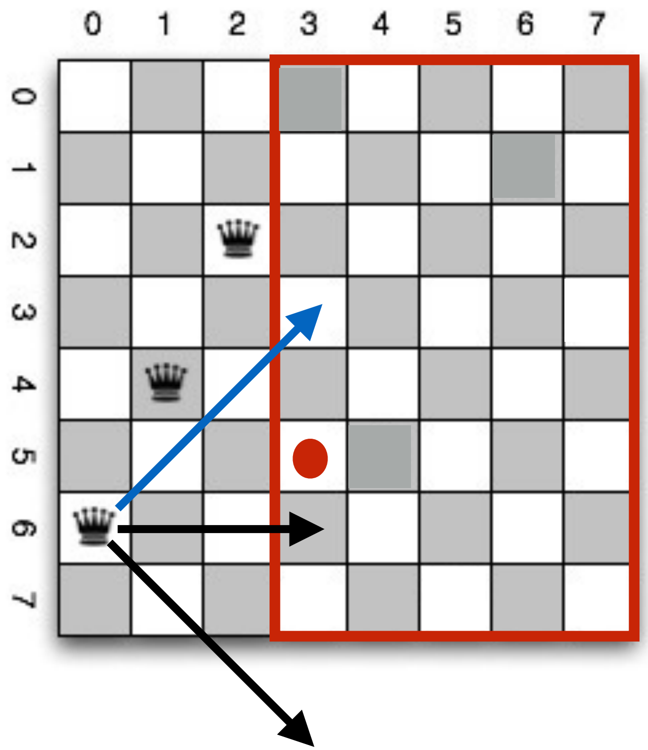

```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
             (row == crow + shift) or
             (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



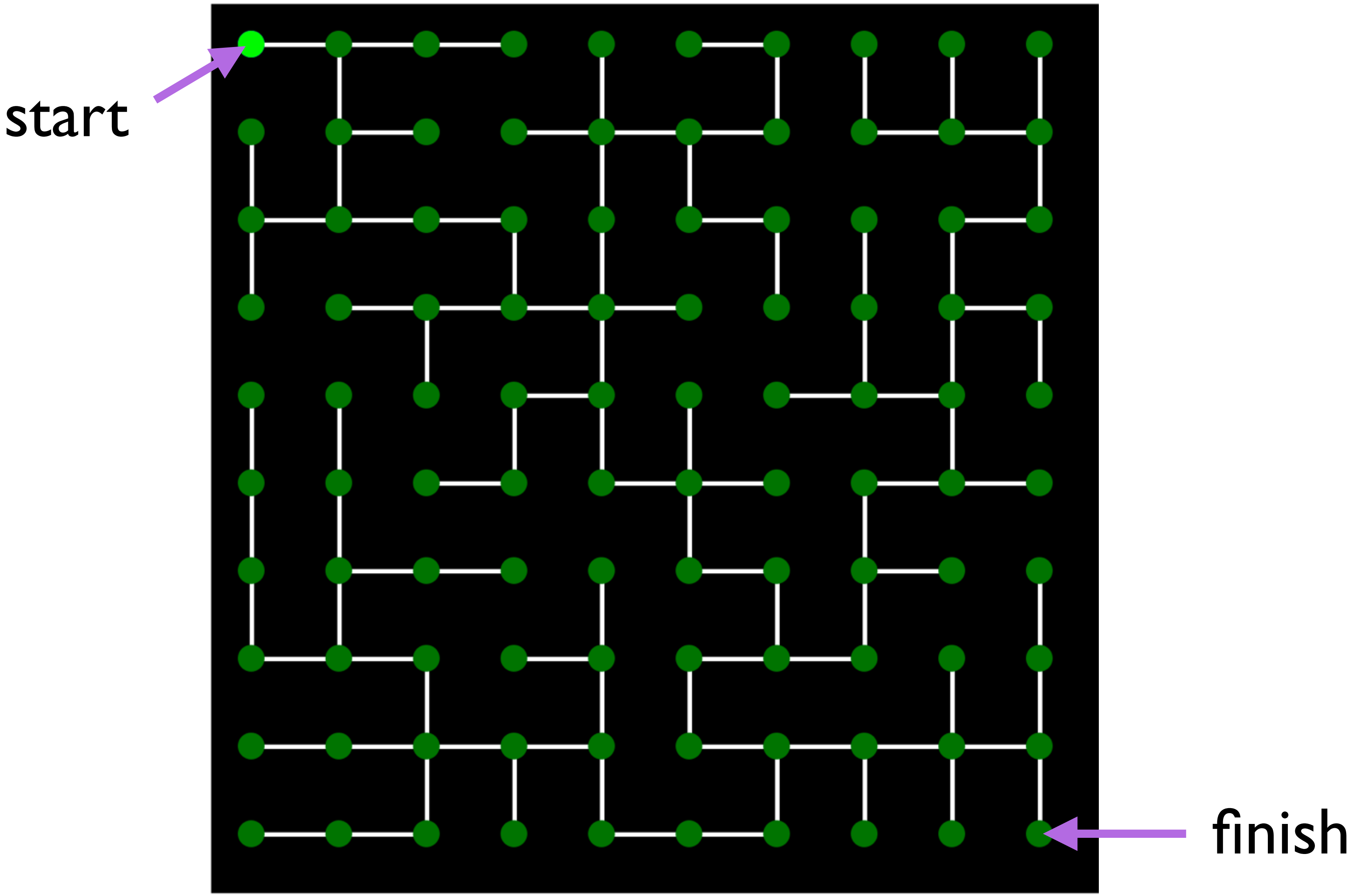

```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
            return False
    return True
```

n = 8

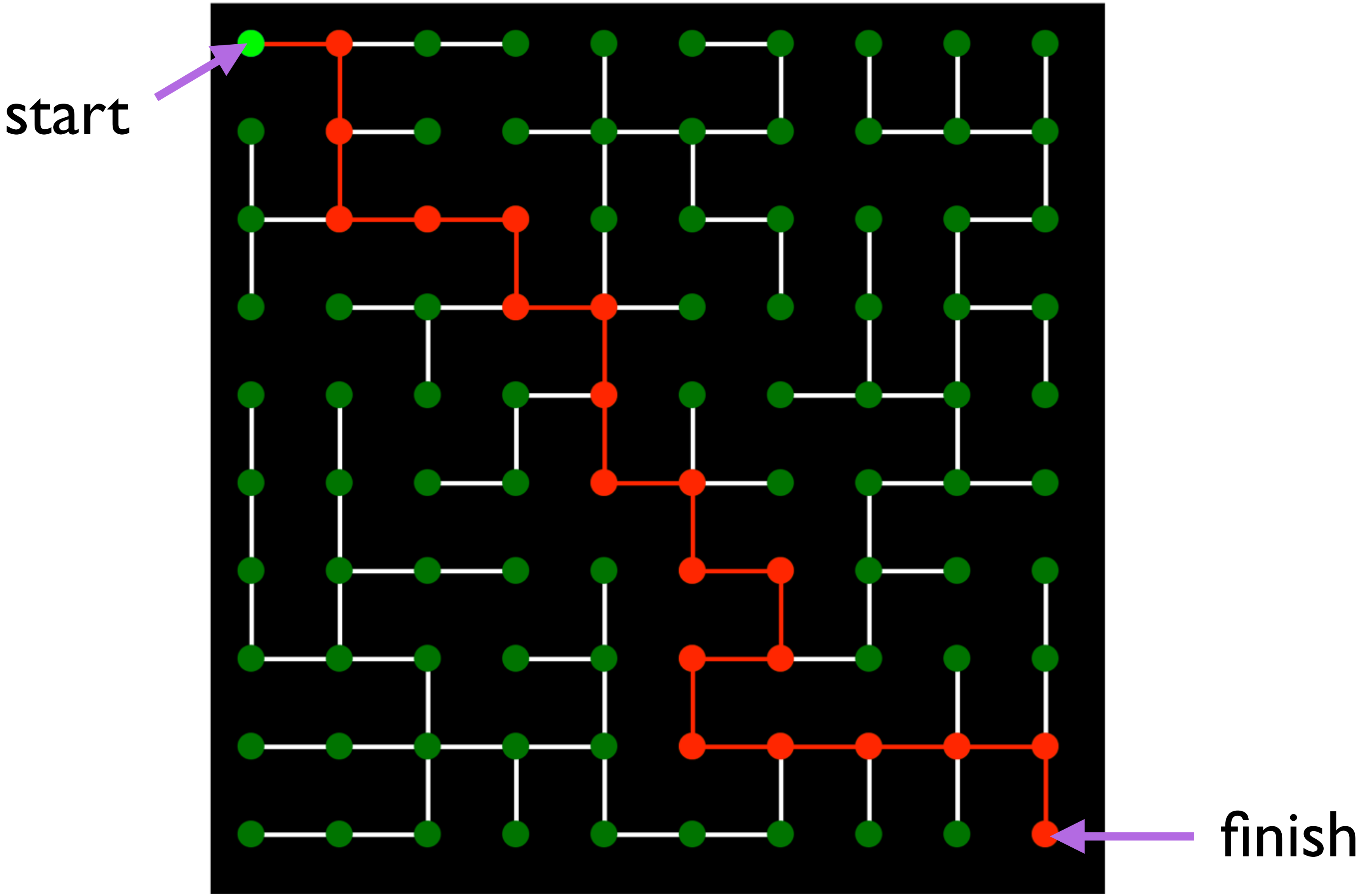
m = 5

constraints = [6,4,2]

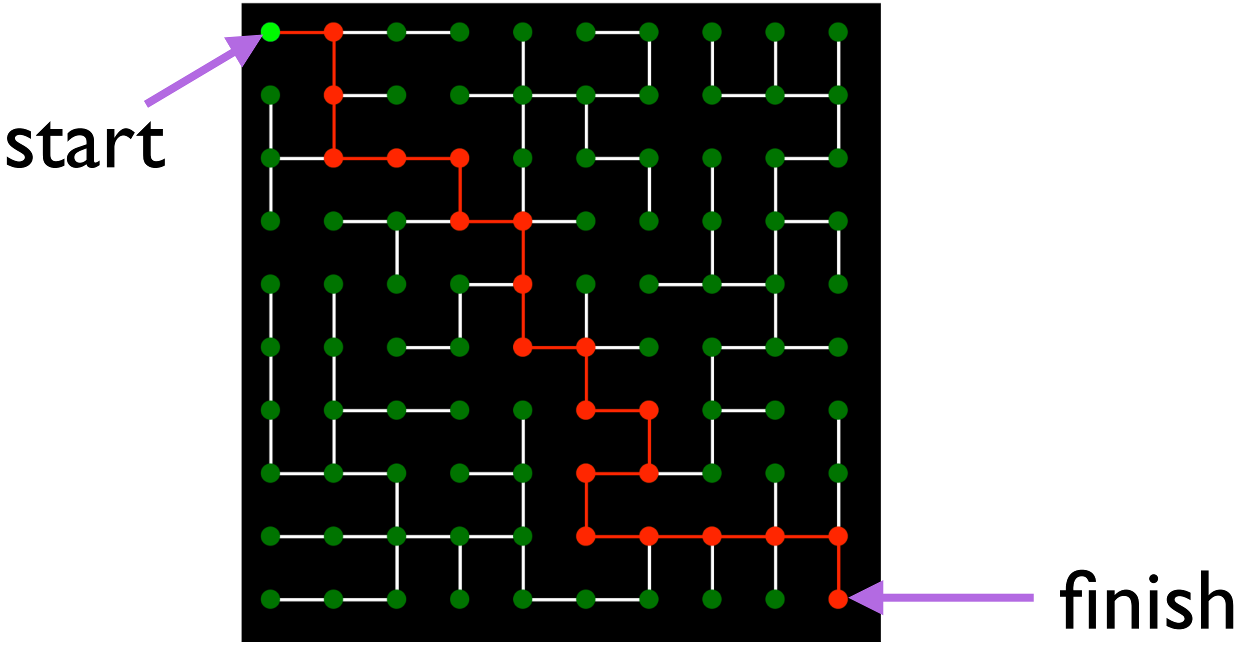
Solving a maze puzzle



Solving a maze puzzle



Solving a maze puzzle



def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):

—> True or False

<u>Main Idea:</u>

if isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)),

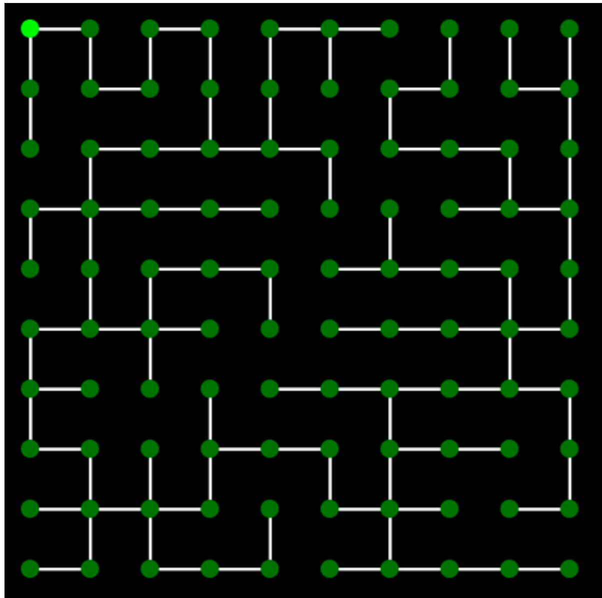
then for some neighbor (rowN, colN) of (rowStart, colStart),

isSolvable(maze, (rowN, colN), (rowEnd, colEnd))

Solving a maze puzzle

```
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True
                  U       D      R      L
    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and
            isSolvable(maze, newCell, (rowEnd, colEnd))):
            return True
    return False
```

Where is the bug?



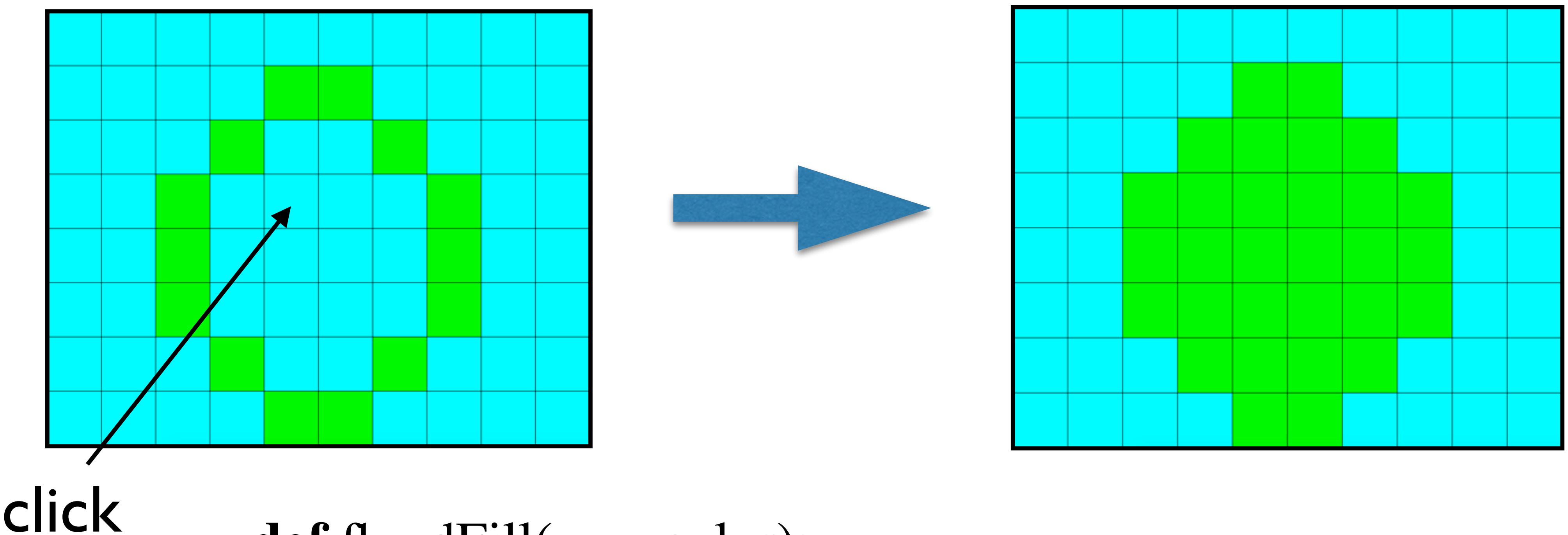
Solving a maze puzzle

```
visited = set()
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) in visited):
        return False
    visited.add((rowStart, colStart))
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True
    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and
            isSolvable(maze, newCell, (rowEnd, colEnd))):
            return True
    return False
```

Solving a maze puzzle

```
visited = set()        solution = set()
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) in visited):
        return False
    visited.add((rowStart, colStart))
    solution.add((rowStart, colStart))
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True
    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and
                isSolvable(maze, newCell, (rowEnd, colEnd))):
                return True
    solution.remove((rowStart, colStart))
    return False
```

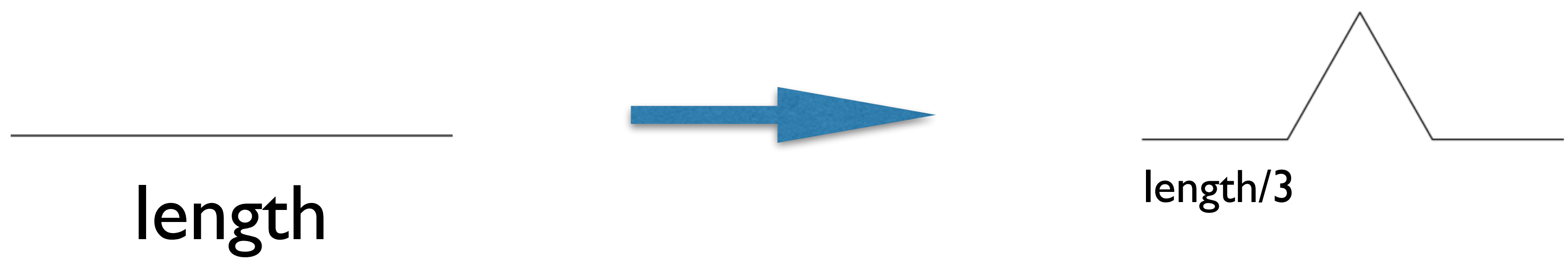
Flood fill




```
def floodFill(x, y, color):
    if ((not inImage(x,y)) or (getColor(img, x, y) == color)):
        return
    img.put(color, to=(x, y))
    floodFill(x-1, y, color)     U
    floodFill(x+1, y, color)     D
    floodFill(x, y-1, color)     L
    floodFill(x, y+1, color)     R
```

Fractals

A change rule:

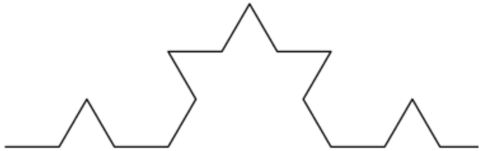


Fractals: kochSnowflake

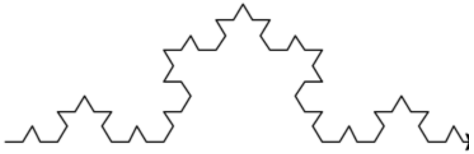
n = 1

n = 2

n = 3

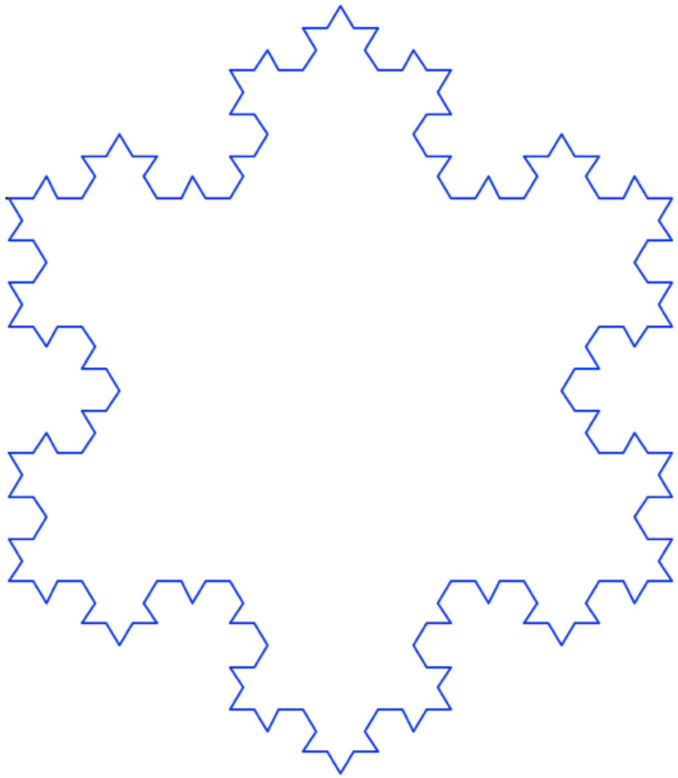


n = 4



```
def kochSide(length, n):
    if (n == 1):
        turtle.forward(length)
    else:
        kochSide(length/3, n-1)
        turtle.left(60)
        kochSide(length/3, n-1)
        turtle.right(120)
        kochSide(length/3, n-1)
        turtle.left(60)
        kochSide(length/3, n-1)
```

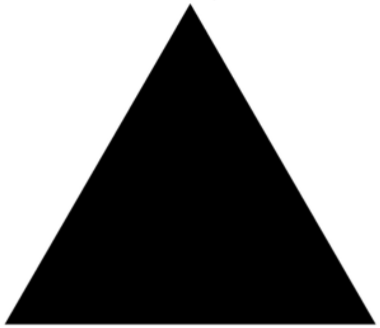
Fractals: kochSnowflake



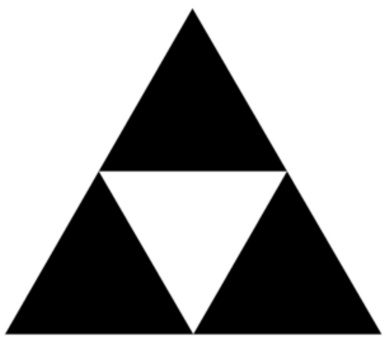
```
def kochSnowflake(length, n):
    # just call kochSide 3 times
    for step in range(3):
        kochSide(length, n)
        turtle.right(120)
```

Fractals: Sierpinski Triangle

level 0



level 1



```
def drawST(x, y, size, level):
  # (x, y) is the bottom-left corner of the triangle
  if (level == 0):
      canvas.create_polygon((x, y),
                            (x+size, y),
                            (x+size/2, y-size*(3**0.5)/2),
                            fill="black" )
  else:
      drawST(x, y, size/2, level-1)
      drawST(x+size/2, y, size/2, level-1)
      drawST(x+size/4, y-size*(3**0.5)/4, size/2, level-1)
```

level 2