Lecture 12.5: More About Recursion

CS2504/CS4092– Algorithms and Linear Data Structures

Dr Kieran T. Herley

Department of Computer Science
University College Cork

2013/14

Summary

The N-Queens problem

The Problem

Problem Position n queens on an $n \times n$ chessboard so that no two queens attack one another

A Solution ($n = 8$)

	0	1	2	3	4	5	6	7
0								Q
1				Q				
2	Q							
3			Q					
4						Q		
5		Q						
6							Q	
7					Q			

May be many solutions (92 for $n = 8$)

The Rules

A queen positioned on a square "attacks" every square that lies on the same row, column, diagonal or anti-diagonal as that square

	0	1	2	3	4	5	6	7
0				.				.
1	.			.			.	
2		.		.		.		
3			.	.	.			
4	.	.	.	Q	.	.	.	.
5			.	.	.			
6		.		.		.		
7	.			.			.	

An Approach

Notation Let $(\mathcal{A}, r)$ represent a legal (non-conflicting) placement of r queens on rows $0 \cdots r-1$

Idea To generate all solutions consistent with arrangement $(\mathcal{A}, r)$:

- If $r = n$, then print solution
- Else try to extend $(\mathcal{A}, r)$ by placing piece on next row (r)
  - For each square not attacked by queen on earlier row,
  - *Tentatively* place queen there giving arrangement $(\mathcal{A}', r+1)$
  - Generate all solutions consistent with $(\mathcal{A}', r+1)$ (recursively)

Illustration

	0	1	2	3	4	5	6	7
0	Q							
1					Q			
2								Q
3	x		x	x	x		x	x
4								
5								
6								
7								

Representing A (Partial) Arrangement

- Note: at most one queen per row/column
- Integer array:

```
board[]          // board[k] gives column number of queen on row k
numPlaced        // no. of queens in current tentative arrangement
```

- Example

	0	1	2	3	4	5	6	7
0	Q							
1					Q			
2								Q
3	x		x	x	x		x	x
4								
5								
6								
7								

board : 0 4 7 ? ? ? ? ?

numPlaced : 3

The Algorithm

```
//
// Generate all solutions consistent with partial arrangement
// contained in board[0..row-1]
//
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens-1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)


GenerateSolns(board, 0, numQueens)
```

Helper Methods

Print Print out the arrangement

Attacked(board, row, col)

- Check if square (row, col) is attacked by any queen on earlier rows
- Return Boolean result

An Algorithm

```
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens−1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)
```

An Algorithm

```
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens−1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)
```

Observation

Algorithm never places queen on square attacked by queen on earlier row

An Algorithm

```
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens−1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)
```

Observation

Algorithm never places queen on square attacked by queen on earlier row

Observation

Any solution printed must be legal

Reasoning About Algorithm

Proposition

GenerateSolns($\mathcal{B}$, r, n) prints all solutions to the n-queens problem consistent with the arrangement represented by $(\mathcal{B}, r)$

Proof.

Use induction on $n - r$ (no. of queens yet to be placed)

Base Case ($n - r = 0$)

```
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens−1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)
```

- Arrangement $(\mathcal{B}, r)$ must be legal by earlier observation
- Since $n = r$, this involves n queens and is a solution
- (Only one consistent with $(\mathcal{B}, r)$)

□

Reasoning About Algorithm

Inductive Case ($n - r > 0$)

```
Algorithm GenerateSolns(board, row, numQueens):
    if row = numQueens then
        Print(board, numQueens)
    else
        for col ←0 to numQueens−1 do
            if not Attacked(board, row, col) then
                board[row] = col
                GenerateSolns(board, row+1, numQueens)
```

- Partial arrangement $(\mathcal{B}, r)$ represented by arguments must be legal
- Any solution consistent with this must have a queen on row r
- Algorithm considers each unconflicted square (r, c), and uses GenerateSolns(B', r+1, n) to print all solutions consistent with $(\mathcal{B}', r+1)$ (if any) where $(\mathcal{B}', r+1)$ represents $(\mathcal{B}, r)$ extended with a queen on square (r, c))
- Clearly this covers all the solutions consistent with $(\mathcal{B}, r)$

Recursion Tree