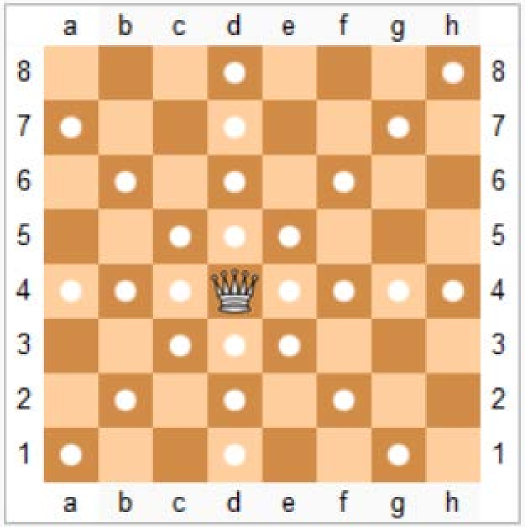


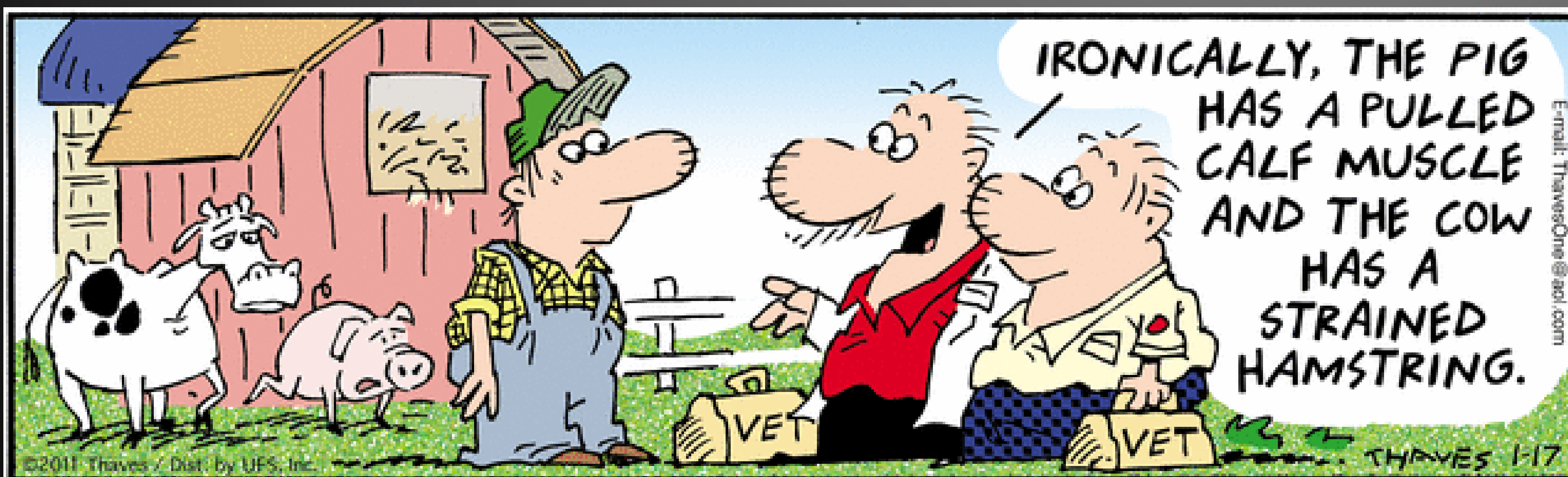
Possible moves of the queen are shown

CSSE 230 Day 16

Exhaustive search, backtracking, object-oriented Queens

Check out from SVN:
Queens

Questions


IRONICALLY, THE PIG HAS A PULLED CALF MUSCLE AND THE COW HAS A STRAINED HAMSTRING.
VET
VET
THAVES 1-17
©2011 Thaves / Dist. by UFS, Inc.
E-mail: ThavesOne@aol.com

Announcements and Agenda

- Exhaustive search, backtracking, and object-oriented queens

Exhaustive Search and Backtracking

A taste of artificial intelligence

Check out Queens from SVN

Exhaustive search

- Given: a (large) set of possible solutions to a problem

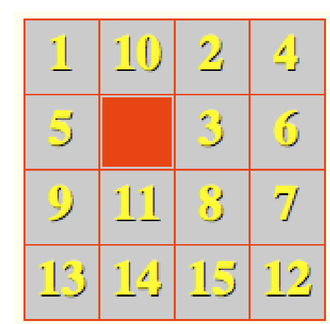
The "search space"

- Goal: Find all solutions (or an optimal solution) from that set

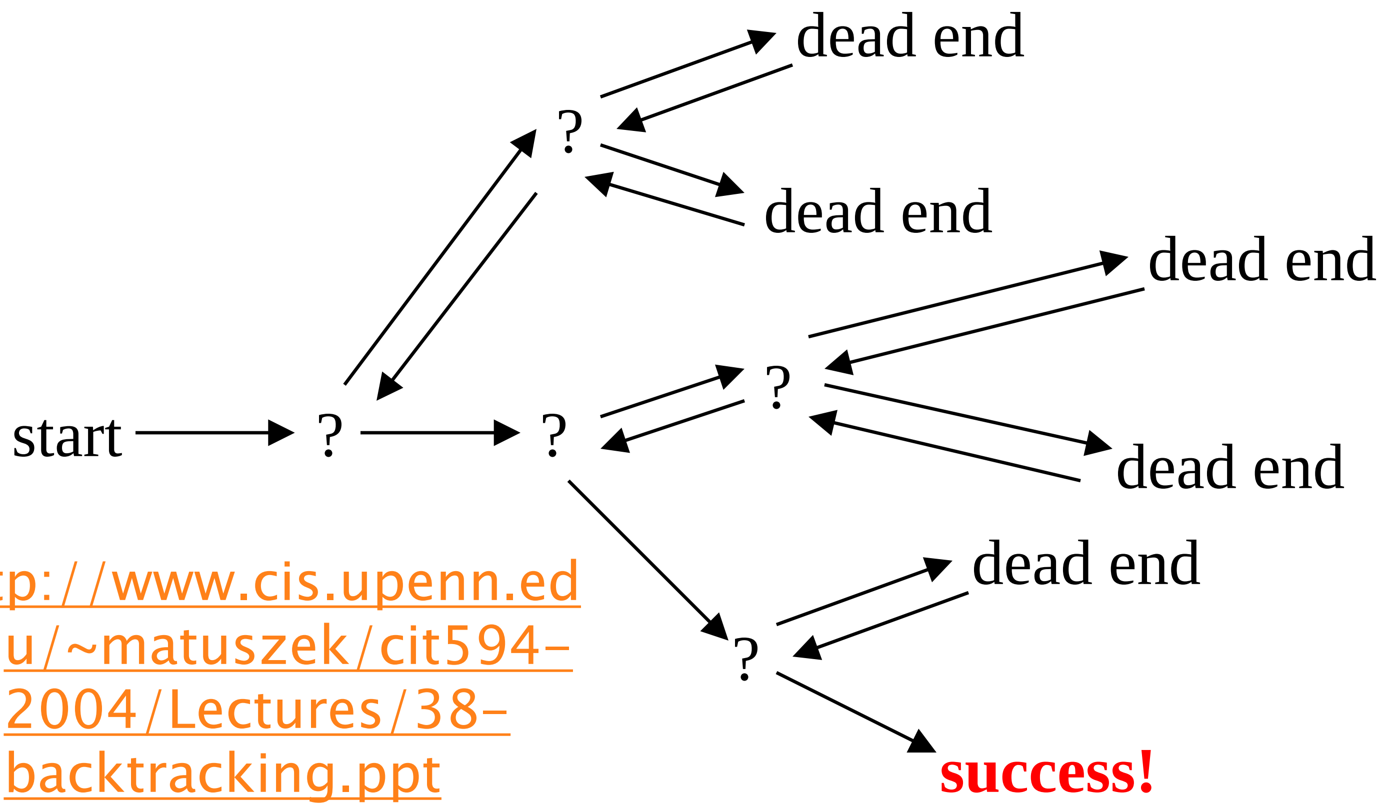
- Questions we ask:
  - How do we represent the possible solutions?
  - How do we organize the search?
  - Can we avoid checking some obvious non-solutions?

In backtracking, we always try to extend a partial solution

- Examples: Doublets, solving a maze, the “15” puzzle.
- Taken from:
    - http://www.cis.upenn.edu/~matuszek/cit594-2004/Lectures/38-backtracking.ppt



In backtracking, we picture the search as a tree and explore it using a pre-order traversal



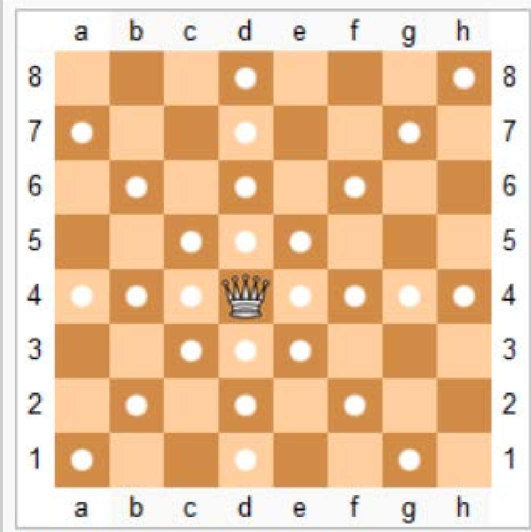
http://www.cis.upenn.edu/~matuszek/cit594-2004/Lectures/38-backtracking.ppt

The non-attacking chess queens problem is a famous example

- In how many ways can N chess queens be placed on an NxN grid, so that none of the queens can attack any other queen?
- I.e. there are not two queens on the same row, same column, or same diagonal.

- There is no "formula" for generating a solution.

- The famous computer scientist Niklaus Wirth described his approach to the problem in 1971: **Program Development by Stepwise Refinement** http://sunnyday.mit.edu/16.355/wirth-refinement.html#3



Possible moves of the queen are shown

http://en.wikipedia.org/wiki/Queen_(chess)

With a partner, discuss "possible solution" search strategies

- In how many ways can N chess queens be placed on an NxN grid, so that none of the queens can attack any other queen?
  - I.e. no two queens on the same row, same column, or same diagonal.

Two minutes
No Peeking!

Search Space Possibilities 1/5

- **Very naive approach. Perhaps stupid is a better word!**
  There are N queens, N^2 squares.
- For each queen, try every possible square, allowing the possibility of multiple queens in the same square.
  - Represent each potential solution as an N-item array of pairs of integers (a row and a column for each queen).
  - Generate all such arrays (you should be able to write code that would do this) and check to see which ones are solutions.
  - Number of possibilities to try in the NxN case:
  - Specific number for N=8:

281,474,976,710,656

Search Space Possibilities 2/5

Slight improvement. There are N queens, N^2 squares. For each queen, try every possible square, notice that we can't have multiple queens on the same square.

- Represent each potential solution as an N-item array of pairs of integers (a row and a column for each queen).
- Generate all such arrays and check to see which ones are solutions.
- Number of possibilities to try in NxN case:
- Specific number for N=8:

178,462,987,637,760

(vs. 281,474,976,710,656)

Search Space Possibilities 3/5

- **Slightly better approach.** There are N queens, N columns. If two queens are in the same column, they will attack each other. Thus there must be exactly one queen per column.
- Represent a potential solution as an N-item array of integers.
  - Each array position represents the queen in one column.
  - The number stored in an array position represents the row of that column's queen.
  - Show array for 4x4 solution.
    - Generate all such arrays and check to see which ones are solutions.
    - Number of possibilities to try in NxN case:
    - Specific number for N=8:

16,777,216

Search Space Possibilities 4/5

- **Still better approach** There must also be exactly one queen per row.

- Represent the data just as before, but notice that the data in the array is a set!
  - Generate each of these and check to see which ones are solutions.
  - **How to generate?** A good thing to think about.
  - Number of possibilities to try in NxN case:
  - Specific number for N=8:

40,320

Search Space Possibilities 5/5

- **Backtracking solution**
- Instead of generating all permutations of N queens and checking to see if each is a solution, we generate "partial placements" by placing one queen at a time on the board
- Once we have successfully placed k<N queens, we try to *extend* the partial solution by placing a queen in the next column.
- When we extend to N queens, we have a solution.

Experimenting with 8 x 8 Case

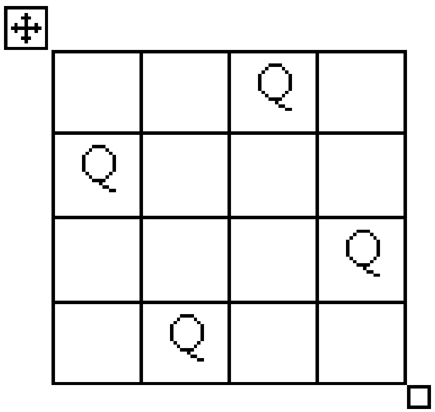
- Play the game:
  - http://homepage.tinet.ie/~pdpals/8queens.htm
- See the solutions:
  - http://www.dcs.ed.ac.uk/home/mlj/demos/queens

Program output:

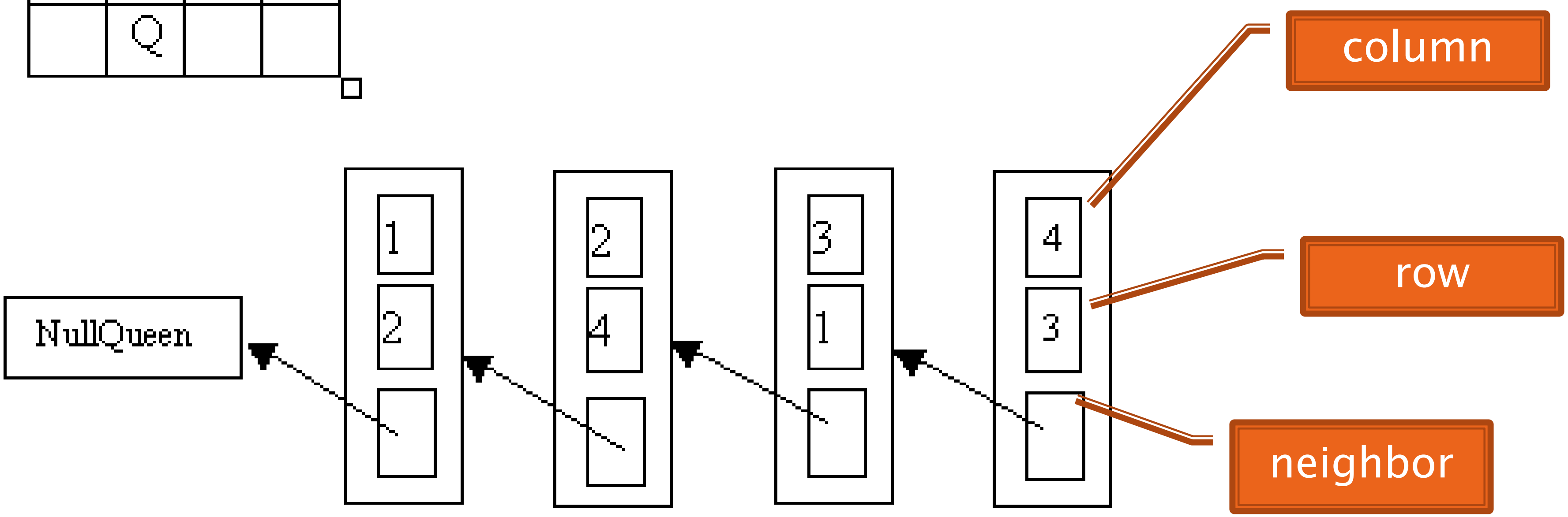
```
>java RealQueen 5
SOLUTION:  1 3 5 2 4
SOLUTION:  1 4 2 5 3
SOLUTION:  2 4 1 3 5
SOLUTION:  2 5 3 1 4
SOLUTION:  3 1 4 2 5
SOLUTION:  3 5 2 4 1
SOLUTION:  4 1 3 5 2
SOLUTION:  4 2 5 3 1
SOLUTION:  5 2 4 1 3
SOLUTION:  5 3 1 4 2
```

The Linked List of Queen Objects

- Board configuration represented by a linked list of ***Queen*** objects



Fields of **RealQueen**:



Designed by Timothy Budd
http://web.engr.oregonstate.edu/~budd/Books/oopintro3e/info/slides/chap06/java.htm

Outline of the algorithm

- Each queen sends messages directly to its immediate neighbor to the left (and recursively to all of its left neighbors)
- Return value provides information concerning **all** of the left neighbors:
- Example: `neighbor.canAttack(currentRow, col)`
  - Message goes to the immediate neighbor, but the real question to be answered by this call is
  - "Hey, neighbors, can any of you attack me if I place myself on this square of the board?"

Queen Methods

- *findFirst()*
- *findNext()*
- *canAttack(int row, int col)*

Your job (part of WA6):

Understand the job of each of these methods.

Javadoc from the Queen interface can help

Fill in the (recursive) details in the RealQueen class

Debug

More details on next slide

More algorithm outline

1. Queen asks its neighbors to find the first position in which none of them attack each other
   - Found? Then queen tries to position itself so that it cannot be attacked.
2. If the rightmost queen is successful, then a solution has been found! The queens cooperate in recording it.
3. Otherwise, the queen asks its neighbors to find the next position in which they do not attack each other
4. When the queens get to the point where there is no next non-attacking position, all solutions have been found and the algorithm terminates

Worktime for Queens problem now

- I will introduce the term project at the end of class

EditorTrees project

Brief description

Meet your team

Reminder: Grading of Team Projects

- I'll assign an overall grade to the project
- Grades of individuals will be adjusted up or down based on team members' assessments

- At the end of the project each of you will:
  - Rate each member of the team, including yourself
  - Write a short Performance Evaluation of each team member with evidence that backs up the rating
    - Positives
    - Key negatives

How to Write a Performance Evaluation

- Document dates and actions:
  - Jan. 1, stayed after mtg. to help Bob with hashing
  - Jan. 19, failed to complete UML diagram as agreed
  - [this means *take notes* so you don't forget]
- List positives:
  - The only way to help people improve!
- List **key** negatives:
  - Not **all** negatives
  - Egos are too fragile for long lists, can't fix everything at once anyway

EditorTrees project

- In general, *implementation* of a Data Structure is separate from *application*.
- Most CSSE 230 projects have used existing data structures to create an application
- In this project you will create an efficient data structure that *could* be used in a text editor.
- But you will *not* create an application that uses it.
- **EditTree:** A height-balanced (but not AVL) binary tree with rank. Insertion and deletion are by **position**, not by natural ordering of the inserted elements.
  - So is it a BST?
- Lots of O(log N) operations
- Uses balance code *and* rank.

EditorTrees Teams

- See schedule page

Meet your partners to plan when you will meet to begin work.

Suggestion: Meet before next class to discuss the project requirements. Formulate a list questions to ask during next class.

Whether or not you meet before next class, read the EditorTrees requirements and come with questions.

Next class will be a workday.