Exhaustive Search

Backtracking
Non-attacking chess queens

Exhaustive search

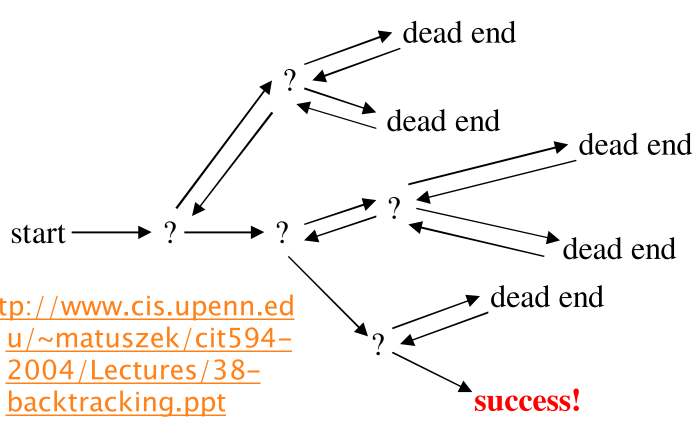
- Given: a (large) set of possible solutions to a problem. Search Space
- Goal: Find all solutions (or an optimal solution) from that set.
  - Is there a way to …
  - List all possible …
  - How many …
- Questions:
  - How do we represent the possible solutions?
  - How do we organize the search?
  - Can we eliminate subsets of the possible solution set without checking each one?

Backtracking

- Always try to extend a partial solution.
- Examples: solving a maze, the "15" puzzle.
- Taken from:
  - http://www.cis.upenn.edu/~matuszek/cit594-2004/Lectures/38-backtracking.ppt

1	10	2	4
5		3	6
9	11	8	7
13	14	15	12

Backtracking (animation)



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

A famous exhaustive search problem

- **Non-attacking chess queens problem**:
  - 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. So we must generate various placements of queens on the board and determine which ones are actually solutions.
- We explore various possibilities for the search space and count the number of potential solutions that must be tried in each case.

4 x 4 solution …

Non-attacking chess queens problem

- In how many ways can *N* chess queens be placed on an *N*x*N* 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.
- In pairs, discuss "possible solution" search strategies (3 minutes).

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 _____________.
  - 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.
- **Demonstrate for the 8x8 case using the applet whose link is on the next slide.**

8 x 8 Case

http://homepage.tinet.ie/~pdpals/8queens.htm

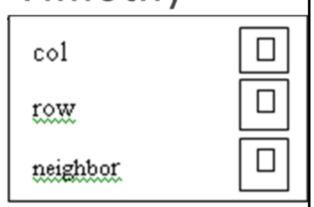
And here is a nice applet showing the solutions:

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
```

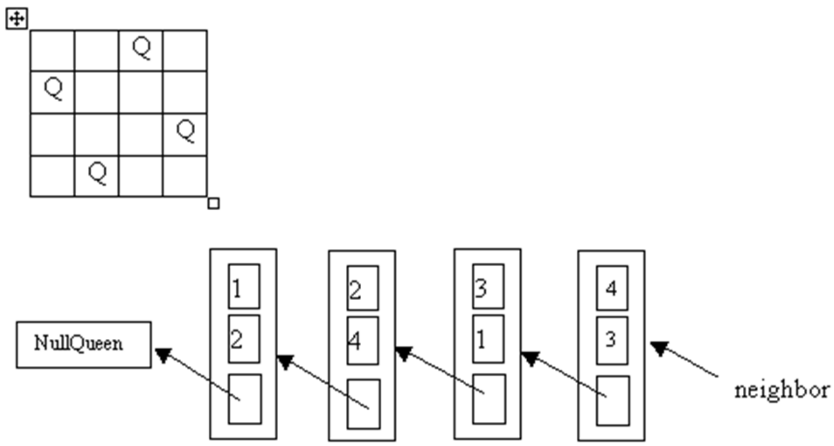
Object-oriented Solution by Timothy Budd



- The queen in each column is represented by a **RealQueen** object.
- Each **RealQueen** knows its column number (fixed), row number (varies), and the queen that is its neighbor to the left (fixed).
- The neighbor of the **RealQueen** in column 1 is a special **NullQueen** object
  - whose purpose is to simplify the code for the **RealQueen** methods
  - by eliminating the need for *if*s that check to see whether a Queen has a neighbor (every **RealQueen** does have a non-null neighbor).

The Linked List of Queen Objects

A board position is represented as a linked list of Queen objects:



Outline of the algorithm

- Each queen sends messages directly to its immediate neighbor to the left, and indirectly to *all* of its left neighbors.
- The return value that this queen receives after sending a message always provides information concerning *all* of the left neighbors.
  For example, when a queen executes
  neighbor.canAttack(currentrow, col);
  The 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?"
- Calls to findFirst() and findNext() have a similar protocol.

More algorithm outline

- A queen asks its neighbors (in the columns to its left) to find the first position in which none of them attack each other.
  - If they can find such a position, this queen tries to position itself so that it does not attack any of its neighbors.
- If the rightmost queen (head of the linked list of queens) is successful at this, a solution has been found, and the queens cooperate in recording it.
- Otherwise, the queen asks its neighbors to find the next position in which they do not attack each other.
- When the queens get to the point where there is no next non-attacking position, all solutions have been found and the algorithm terminates.