CompSci 100e
Program Design and Analysis II

March 3, 2011
Prof. Rodger

Announcements
- Written assignment due tonight – put .pdf file in Eclipse and submit it
- APTS due Tuesday after break – use recursion
- No Lab Friday March 4 or Monday March 14
- Today – two examples of recursion
  - Blob counting
  - 8 Queens - backtracking

Blob Counting, Flood Fill
- Flood a region with color
  - Erase region, make transparent, ..
  - How do find the region?

- Finding regions, blobs, edges, ..
  - See blob counting code
  - What is a blob?

- Recursion helps, but necessary?
  - Performance, clarity, ...
  - Ease of development

First Example - BlobCount
- How do we find images? Components? Paths?
  - Create information from data
Details and Idioms in blob code

- Method `blobFill` has four parameters
  - (row,column) of where search starts
  - Character being searched for (initially * or blob)
  - Character to fill with on success (e.g., count ‘2’ or ‘4’)
    - Mark for visualization
    - Mark to ensure we don’t search again!
- If (row,column) is part of blob, count it and ask neighbors for their counts
  - They’re part of blob (if never visited before)
- Return total of yourself and neighbors
  - Key to recursion: do one thing and ask for help

Blob questions

- What changes if diagonal cells are adjacent?
  - Conceptually and in code

- How do we find blob sizes in a range?
  - Not bigger than X, but between X and Y

- How would we number blobs by size rather than by when they’re found?
  - Do we have the tools to do this in existing code?

- Can we avoid recursion and do this iteratively?

Second Example – 8 Queens

Backtracking by image search
Searching with no guarantees

- Search for best move in automated game play
  - Can we explore every move?
  - Are there candidate moves ranked by “goodness”? 
  - Can we explore entire tree of possible moves?

- Search with partial information
  - Predictive texting with T9 or iTap or ...
  - Finding words on a Boggle board
  - What numbers fit in Sudoku square

- Try something, if at first you don’t succeed ....

Classic problem: N queens

- Can queens be placed on a chess board so that no queens attack each other?
  - Easily place two queens
  - What about 8 queens?
- Make the board NxN, this is the N queens problem
  - Place one queen/column
  - Horiz/Vert/_diag attacks
- Backtracking
  - Tentative placement
  - Recurse, if ok done!
  - If fail, undo tentative, retry
- wikipedia-n-queens

Search, Backtracking, Heuristics

- How do you find a needle in a haystack?
  - How does a computer play chess?
  - Why would you write that program?
  -...
- How does Bing/Googlemap find routes from one place to another?
  - Shortest path algorithms
  - Longest path algorithms

- Optimal algorithms and heuristic algorithms
  - When is close good enough? How do measure “closeness”? 
  - When is optimality important, how much does it cost?

Backtracking idea with N queens

- For each column C, tentatively place a queen
  - Try first row in column C, if ok, move onto next column
    - Typically “move on” is recursive
  - If solved, done, otherwise try next row in column C
    - Must unplace queen when failing/unwind recursion
- Each column C “knows” what row R it’s on
  - If first time, that’s row zero, but might be an attack
  - Unwind recursion/backtrack, try “next” location
- Backtracking: record an attempt go forward
  - Move must be “undoable” on backtracking/unwinding
N queens backtracking: Queens.java

```java
public boolean solve(int col){
    if (col == mySize) return true;
    // try each row until all are tried
    for(int r=0; r < mySize; r++){
        if (myBoard.safeToPlace(r, col)){
            myBoard.setQueen(r, col, true);
            if (solve(col+1)){
                return true;
            }
        }
    }
    myBoard.setQueen(r, col, false);
    return false;
}
```

Basic ideas in backtracking search

- Enumerate all possible choices/moves
  - We try these choices in order, committing to a choice
  - If the choice doesn’t pan out we must undo the choice
  - Backtracking step, choices must be undoable

- Inherently recursive, when to stop searching?
  - When all columns tried in N queens
  - When we have found the exit in a maze
  - When every possible moved tried in Tic-tac-toe or chess?
    - Is there a difference between these games?

- Summary: enumerate choices, try a choice, undo a choice, this is **brute force** search: try everything

Pruning vs. Exhaustive Search

- If we consider every possible placement of 4 queens on a 4x4 board, how many are there? (N queens)
  - 4x4x4x4 if we don’t pay attention to any attacks
  - 4x3x2x1 if we avoid attacks in same row

- What about if we avoid diagonal attacks?
  - Pruning search space makes more search possible, still could be lots of searching to do!

- Estimate how long to calculate # solutions to the N-queens problem with our Java code....

Queens Details

- How do we know when it’s safe to place a queen?
  - No queen in same row, or diagonal
  - For each column, store the row that a queen is in
  - See QBoard.java for details

- For GUI version, we use a **decorator**
  - The QBoardGUI is an IQueenState class and it has an IQueenState object in it
  - Appears as an IQueenState to client, but uses an existing one to help do its work
  - One of many object oriented design patterns, seen in Huff in the BitInputStream class