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
A page from a document discussing blob tracking. The page contains text and diagrams illustrating the concept, with questions posed in the text.

**Questions:**
- Can we avoid recursion and do this iteratively?
- Do we have the tools to do this in existing code?
- How would we number blobs by size rather than by position?
- How do we find blob sizes?
- Conceptually and in code:
  - What changes if diagonal cells are adjacent?

**Blob Questions:**

**Details and Idioms in Blob Code:**

**Second Example – 8 Queens:**

**Key to recursion:**
- Do one thing and ask for help.
- Return total of your self and neighbors.
- They're part of blob (if never visited before).
- If (row, column) is part of blob, count it and ask neighbors for their counts.
- Mark for visualization.
- Character to fill with on success (e.g., count 2, or 4?).
- Character before search for (initially * or blob).
- (Row, column) of where search starts.

**Method Blob[][]** has four parameters.
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 ....

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?

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

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
Queens

Queens

Pruning vs. Exhaustive Search

Basic Ideas in Backtracking Search

Java

Queens Backtracking:

```java
public boolean solveNQueens(int n) {
    return solveNQueens(n, 0);
}

private boolean solveNQueens(int n, int row) {
    if (row == n) {
        return true;
    }

    for (int col = 0; col < n; col++) {
        if (isSafe(row, col)) {
            if (solveNQueens(n, row + 1)) {
                return true;
            }
        }
    }

    return false;
}
```

Overview of the Queens Problem:
- The problem involves placing N queens on an N×N chessboard so that no two queens threaten each other.
- Each queen can move horizontally, vertically, or diagonally.
- The goal is to find all possible configurations where no queen threatens another.

Java Implementation:
- The `solveNQueens` method recursively tries to place queens row by row.
- `isSafe` checks if a queen can be placed at a given position without being attacked by any other queen.

Summary:
- Enumerate choices, try a choice, undo a choice.
- There is a difference between these games.
- When every possible move is tried in a tic-tac-toe or chess board.
- When we have found the next legal position.
- When all columns tried in a Queens.
- Inherently recursive, when to stop searching?

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

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

Could be lots of searching to do.
- Pruning search space makes more search possible, still
- What about if we avoid diagonal attacks?
- 4×4×2^4 if we avoid attacks in some row
- 4×4×4×4 if we don't pay attention to any attacks
- 4×4 board, how many are there? (N queens)
- If we consider every possible placement of 4 queens on

Queens Details

See Qboard.java for details.

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

Pruning vs. Exhaustive Search

Summary: enumerate choices, try a choice, undo a choice.

Is there a difference between these games?
- When every possible move is tried in a tic-tac-toe or chess board.
- When we have found the next legal position.
- When all columns tried in a Queens.
- Inherently recursive, when to stop searching?

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