Backtracking, Search, Heuristics

- Many problems require an approach similar to solving a maze
  - Certain mazes can be solved using the “right-hand” rule
  - Other mazes, e.g., with islands, require another approach
  - If you have “markers”, leave them at intersections, don’t explore the same place twice

- What happens if you try to search the web, using links on pages to explore other links, using those links to ...
  - How many web pages are there?
  - What rules do web crawlers/web spiders follow?
    - Who enforces the rules?

- Keep track of where you’ve been don’t go there again
  - Any problems with this approach?
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 N×N, this is the N queens problem
  - Place one queen/column
  - # different tries/column?

- Backtracking
  - Use “current” row in a col
  - If ok, try next col
  - If fail, back-up, next row
Backtracking idea with N queens

- **Try to place a queen in each column in turn**
  - Try first row in column C, if ok, move onto next column
  - If solved, great, otherwise try next row in column C, place queen, move onto the next column
    - Must unplace the placed queen to keep going

- **What happens when we start in a column, where to start?**
  - If we fail, move back to previous column (which remembers where it is/failed)
  - When starting in a column anew, start at beginning
    - When backing up, try next location, not beginning

- **Backtracking in general, record an attempt go forward**
  - If going forward fails, undo the record and backup
Basic ideas in backtracking search

- We need to be able to 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
    • This is the backtracking step, choices must be undoable

- Process is inherently recursive, so we need to know when the search finishes
  - 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
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;
}
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
Computer v. Human in Games

- Computers can explore a large search space of moves quickly
  - How many moves possible in chess, for example?

- Computers cannot explore every move (why) so must use heuristics
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move

- What do humans do well in these games? What about computers?
  - What about at Duke?
Heuristics

- A heuristic is a rule of thumb, doesn’t always work, isn’t guaranteed to work, but useful in many/most cases
  - Search problems that are “big” often can be approximated or solved with the right heuristics

- What heuristic is good for Sudoku?
  - Is there always a no-reasoning move, e.g., 5 goes here?
  - What about “if I put a 5 here, then...”
  - Do something else?

- What other optimizations/improvements can we make?
  - For chess, checkers: good heuristics, good data structures