Programming Abstractions

CS106B

Julie Zelenski (for Cynthia Lee)
Today’s topics:

- Recursion Week
- Fortnight comes to its thrilling conclusion
- Today:
  - Trace/visualize recursion in action
  - Applying backtracking template to new problems
Recap: backtracking template

```cpp
bool backtrackingRecursiveFunction(args) {
    › Base case test for success: return true
    › Base case test for failure: return false
    › Loop over several options for “what to do next”:
      1. Tentatively “choose” one option
      2. if (“explore” with recursive call returns true) return true
      3. else That tentative idea didn’t work, so “un-choose” that option,
         but don’t return false yet!--let the loop explore the other options before giving up!
    › None of the options we tried in the loop worked, so return false
}
```

Stanford University
One template, many applications

- **Navigating a maze**
  - *Goal*: Find path from start to exit
  - *Choose/unchoose*: which direction to move from current location
  - *Base cases*: exited maze, hit dead end

- **Locker lock combinations**
  - *Goal*: Find combination that unlocks
  - *Choose/unchoose*: which number to extend current combination
  - *Base cases*: current combination opens lock, hit dead end

- **Element speller**
  - *Goal*: Divide a word into element substrings
  - *Choose/unchoose*: nibble off prefix from word
  - *Base cases*: remaining word is empty, hit dead end
Recursion exploration as a “decision tree”

- Horizontal branching for loop choices at decision point
  - Wider branching indicates more options
- Vertical level for recursive call
  - Taller tree indicates more decisions to make
- Choose-explore: "optimistic" choice and recurse onward in hopes of solution
- Backtrack: returns to most recent decision point and un-chooses
- Exponential growth
  - If W is number of options, N is number of decisions, exhaustive exploration of tree is \( W^N \)
  - Which parameter contributes more expense: W or N?
- How much of tree must be explored to find a solution? To find all solutions?
  - How deep does function call stack get?
Decision tree for locker combinations

How does above decision tree compare to that for element spelling?

Stanford University
Placing N Queens

- The powerful queen piece can attack on straight line
  - Horizontally, vertically, diagonally
- Goal is to place N queens on a N×N chess board such that no queen threatens any other

- Start with empty board, N queens to place
  - Decision places one queen
  - Recurse to place rest of queens

- What are choices for where to place queen?
  - Is the entire board really up for grabs? How might we refine to only those options that are viable?
- What does decision tree look like?
NQueens code

```cpp
bool solveQueens(Board& board, int row)
{
    if (row == board.size()) {
        return true;
    } else {
        for (int col = 0; col < board.size(); col++) {
            if (board.isSafe(row, col)) {
                board.place(row, col);
                if (solveQueens(board, row + 1)) {
                    return true;
                }
                board.remove(row, col);
            }
        }
        return false;
    }
}
```

1. Choose
2. Explore rest with recursion
3. Unchoose
Sudoku puzzle

- Assign digits to empty cells such that digits 1-9 are used once in each row, each column, and each block.

- Start with N empty cells to assign
  - Each decision assigns a digit to an empty cell
  - Recurse to explore filling rest of puzzle

- Which digits are worth trying? Which are not?
- What does decision tree look like?
### Sudoku code

```cpp
bool solveSudoku(Board& board)
{
    int row, col;

    if (!board.findEmpty(row, col)) {
        return true;
    } else {
        for (int digit = 1; digit <= 9; digit++) {
            if (board.isUnique(row, col, digit)) {
                board.place(row, col, digit);
                if (solveSudoku(board)) {
                    return true;
                }
                board.remove(row, col);
            }
        }
    }
    return false;
}
```

1. Choose
2. Explore rest with recursion
3. Unchoose

Stanford University
Summary: Backtracking in practice

- Problem has recursive, self-similar nature
  - Sequence of decision points
  - Nibble off one decision, recurse on rest
  - Each decision progresses to smaller/simpler version of same problem
- Fit to backtracking template:
  - Choose/unchoose: try all choices for a given decision point, if none work, backtrack
  - Recursion to explore subsequent decisions
  - Base cases: identify success, failure
- Coding mechanics:
  - How to model state of exploration
  - Update/communicate state into and out of recursive calls
  - How to loop/enumerate options
- Refinements to consider:
  - Efficiency: order of exploration, pruning, avoid redundancy
  - Declutter: streamline algorithm by abstracting details into helpers