Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

Today's topics:

- Recursion ~~Week~~ Fortnight comes to its thrilling conclusion!
- Today:
  - Visualizing recursive backtracking as a decision tree
  - Applying our recursive backtracking template to new problems
  - Theme and variations: state speller code example

- Admin
  - Assign 4 out and due this Friday
    - Assignment parade takes a breather, Assign 5 released after diagnostic
  - Mid-quarter diagnostic next week
    - Any 3-hour block within 24-hour window Tue Oct 26th 5pm - Wed Oct 27th 5pm
    - Sample posted on Gradescope later this week

Backtracking template

bool backtrackingRecursiveFunction*(args)* {

- Base case test for success: return true
- Base case test for failure: return false
- Loop over available 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

}

One template, many applications

- **Combination lock**
  - Goal: find combo that unlocks
  - Choose/unchoose: {0-9} which digit to extend combo
  - Base case: combo is full length, does it open lock?
- **Gift card**
  - Goal: spend card down to zero
  - Choose/unchoose: {yes-no} whether to buy item
  - Base cases: no money on gift card, no items left to consider
- **Solve maze**
  - Goal: exit maze
  - Choose/unchoose: {N-S-E-W} which direction to move
  - Base case: found exit

Recursive exploration as "decision tree"

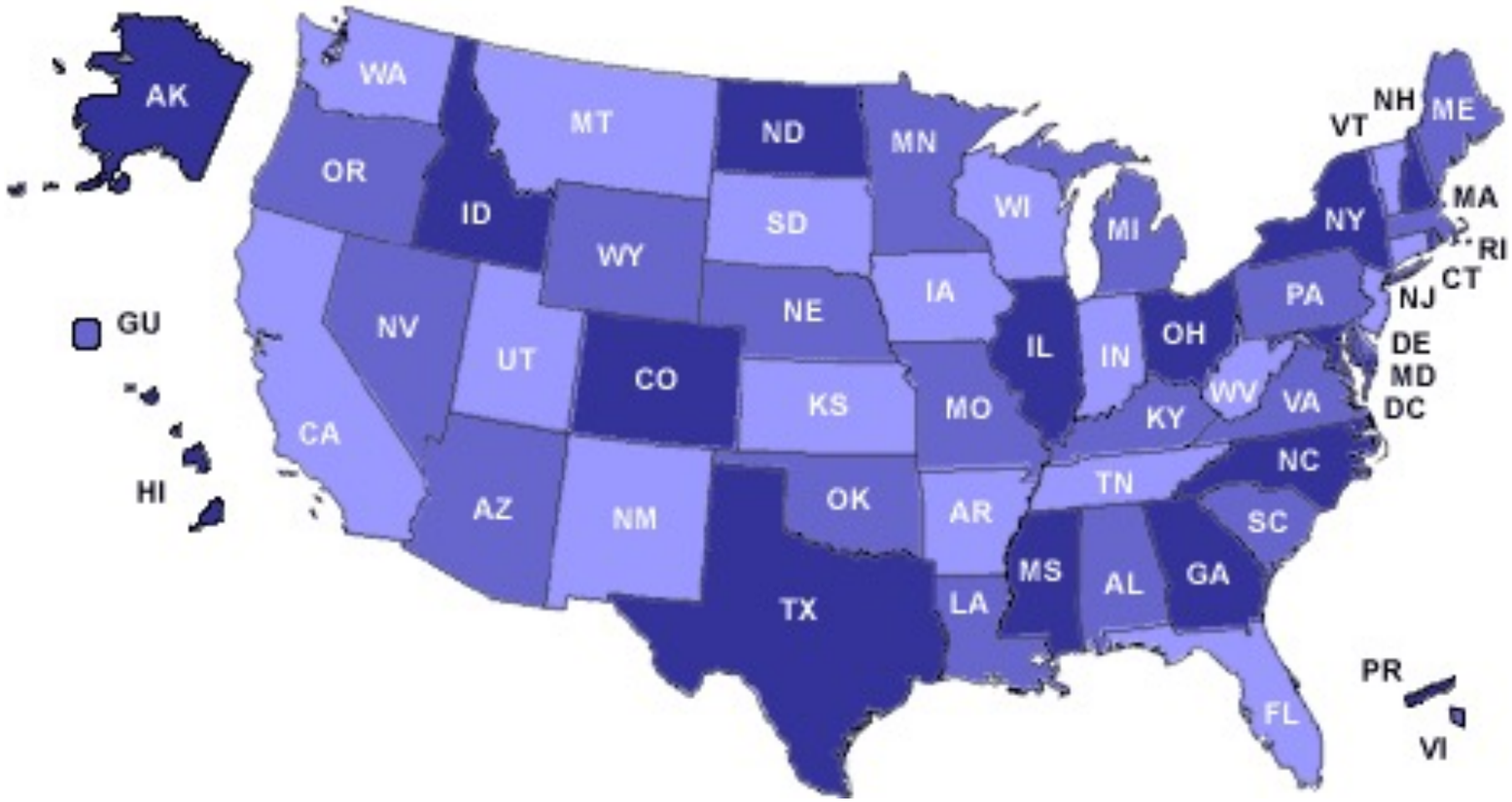
- **Count of horizontal branches at each decision point is *width***
  - More branches = more options to choose from
- **Count of vertical levels is *depth***
  - Taller tree = more decisions to make
- **Exponential growth**
  - If W is count of options and D is count of decisions, exhaustive exploration of entire tree is $O(W^D)$
    - That can be a **lot** of work…!
    - What is impact on performance of larger W? of larger D?
  - How much of tree is explored to find **a** solution? to find **all** solutions?
    - How deep does function call stack get?

Code Example

STATE SPELLER

State speller

Which words can be spelled out of state postal codes? CO + DE = CODE!



State speller

- You are given:
  - `Set<string> state`: postal codes AL, CA, FL, …
  - `Lexicon` of English words
- This first version explores all combos of length n and prints those that are words

```
void printStateWords(int n, Set<string>& states, Lexicon& lex, string sofar)
{
    if (n == 0) {
        if (lex.contains(sofar)) {
            cout << sofar << endl;
        }
    } else {
        for (string option : states) {
            printStateWords(n - 1, states, lex, sofar + option);
        }
    }
}
```

State speller

- **What are some variations we can apply to this code?**
  - What do we change …
  - to print all words of any length
  - to build a set of words
  - to return count of words
  - to prune dead ends (not valid prefix)
  - to allow/disallow repeat of postal codes in word (combos vs permute)
  - to stop at first word found, return true/false
  - to stop at first word found, return word
  - to return longest word found
  - and many others…
- **Let's do this together in Qt!!**

Summary: Recursive backtracking in practice

- **Identify how problem has recursive, self-similar structure**
  - Diagram as decision tree, sequence of decisions is path down tree
  - Nibble off one decision, recurse on rest
  - Each decision progresses to smaller/simpler version of same problem
- **Fit to backtracking template**
  - Base cases: success and failure
  - Choose/explore/unchoose
- **How to model state of exploration**
  - Update/communicate state into and out of recursive calls
  - How to loop/enumerate options
- **Theme and variations**
  - Print all, count, find one, find all, find optimal