CS 42—Optimization, Part 2

Tuesday, November 6, 2018

Summary

Today, we'll continue learning about optimization techniques, in particular **dynamic programming with tabulation**.

Dynamic programming with tabulation

Dynamic-programming with tabulation is an optimization technique that avoids doing redundant work. Dynamic programming uses a table to store the results of subcomputations, and computes smaller versions of the problem before larger versions of the problem.

Here's how dynamic programing works:

1. Design and implement a straightforward algorithm (often as a recursive function *f*).
2. Determine that *f* performs redundant work and can be optimized.
3. Using the intuition you gained from implementing the recursive version, design the dynamic-programming (DP) table, by asking the following questions:
    1. **What is the meaning of a cell?** What kind of information is stored in a cell? (i.e., what *is* a subcomputation)?
    2. **How many cells will there be**, for an input of size N? (i.e., how many unique recursive calls / subproblem solutions are needed to solve the full problem of size N?)
    3. **Which cell contains the answer** to the full problem of size N? (i.e., we'll eventually return the value of which cell?)
    4. **Which cells are easy to fill in**, and how do we do so? (i.e., which cells correspond to base cases, and what are their values?)
    5. **How does the value of a single cell depend on the value of other cells?** (i.e., how do the recursive cases work?)
    6. **In what order should we fill in the cells**, so that we can be sure to compute the results for each subproblem *before* it's needed?
4. Write the code
    1. Consider directly returning values for the base case(s).
    2. To compute the "recursive cases", create a table of the appropriate size.
    3. Write code to fill in the base-case cells.
    4. Write code (usually a loop) to fill in the remaining values. The loop should go in the order you determined above, and each iteration of the loop fills in the value of one cell.
    5. Return the value in the result cell.

Next time: analysis with summations + wrapping up optimization