A ***tabular*** method!

Basics of Dynamic Programming

Dynamic Programming

- Not a specific algorithm, but a technique (like divide-and-conquer).
- Developed back in the day when "programming" meant "tabular method" (like linear programming). Doesn't really refer to computer programming.
- Used for optimization problems (a set of choices must be made in order to arrive at an optimal solution):
  - Find *a* solution with *the* optimal value.
  - Minimization or maximization.

Dynamic Programming

- Another strategy for designing algorithms is *dynamic programming*
  - A metatechnique, not an algorithm (like divide & conquer)
  - The word “programming” is historical and predates computer programming
- Use when problem breaks down into recurring small subproblems

Optimization Problems

- In an optimization problem, there are typically many *feasible* solutions for any input instance I
- For each solution S, we have a *cost* or *value* function f(S)
- Typically, we wish to find a feasible solution S such that f(S) is either *minimized* or *maximized*
- Thus, when designing an algorithm to solve an optimization problem, we must prove the algorithm produces a best possible solution.

Principle of Optimality

- In book, this is termed "Optimal substructure"
- An optimal solution contains within it optimal solutions to subproblems.
- More detailed explanation
  - Suppose solution S is optimal for problem P.
  - Suppose we decompose P into P_1 through P_k and that S can be decomposed into pieces S_1 through S_k corresponding to the subproblems.
  - Then solution S_i is an optimal solution for subproblem P_i

Divide-and-Conquer	Dynamic Programming
Combines solutions of subproblems to solve the original problem	
Disjoint subproblems	***Overlapping*** subproblems

Dynamic Programming

- Dynamic programming is a divide-and-conquer technique at heart
- That is, we solve larger problems by patching together solutions to smaller problems
- Dynamic programming can achieve efficiency by storing solutions to subproblems to avoid redundant computations
  - We typically avoid redundant computations by computing solutions in a bottom-up fashion

Dynamic Programming (DP)

- Like divide-and-conquer, solve problem by combining the solutions to sub-problems.
- Differences between divide-and-conquer and DP:
  - Independent sub-problems, solve sub-problems independently and recursively, (so same sub(sub)problems solved repeatedly)
  - Sub-problems are dependent, i.e., sub-problems share sub-sub-problems, every sub(sub)problem solved just once, solutions to sub(sub)problems are stored in a table and used for solving higher level sub-problems.

8

Optimal Substructure

- Optimal substructure
- Overlapping subproblems

Optimal Substructure – contd.

- Optimal (sub)structure
  - An optimal solution to the problem contains within it optimal solutions to subproblems.
- Overlapping subproblems
  - The space of subproblems is “small” in that a recursive algorithm for the problem solves the same subproblems over and over. Total number of distinct subproblems is typically polynomial in input size.
- (Reconstruction an optimal solution)

Optimal Substructure – contd.

- We say that a problem exhibits **optimal substructure** if an optimal solution to the problem contains within it optimal solution to subproblems.
- Example: Matrix-multiplication problem

Optimal Substructure – contd.

Optimal substructure varies across problem domains in two ways:

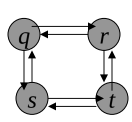
1. how many subproblems are used in an optimal solution to the original problem, and
2. how many choices we have in determining which subproblem(s) to use in an optimal solution.

Subtleties when Determining Optimal Structure

- Take care that optimal structure does not apply even it looks like to be in first sight.
- Unweighted shortest path:
  - Find a path from u to v consisting of fewest edges.
  - Can be proved to have optimal substructures.

Subtleties when Determining Optimal Structure – contd.

- Unweighted longest simple path:
  - Find a simple path from u to v consisting of most edges.
  - Figure 15.4 shows it does not satisfy optimal substructure.
- Independence (no share of resources) among subproblems if a problem has optimal structure.



$q \rightarrow r \rightarrow t$ is the longest simple path from q to t.
But $q \rightarrow r$ is not the longest simple path from q to r.

14

Efficient Top-Down Implementation

- We can implement any dynamic programming solution top-down by storing computed values in the table
  - If all values need to be computed anyway, bottom up is more efficient
  - If some do not need to be computed, top-down may be faster

Rules for Dynamic Programming

1. Characterize the structure of an optimal solution.
2. Derive a recursive formula for computing the values of optimal solutions.
3. Compute the value of an optimal solution in a bottom-up fashion (top-down is also applicable).
4. Construct an optimal solution from computed information.

When is dynamic programming effective?

- Dynamic programming works best on objects that are linearly ordered and cannot be rearranged
  - characters in a string
  - files in a filing cabinet
  - points around the boundary of a polygon
  - the left-to-right order of leaves in a search tree.
- Whenever your objects are ordered in a left-to-right way, dynamic programming must be considered.

The End