Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2025

Dynamic Programming Practice

Simple Example: Fibonacci Numbers

The Fibonacci sequence is defined by:

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0$$
$$F(1) = 1$$

Direct computation of $F(7)$:

If we save the answers to solved subproblems in a table:

Fibonacci bottom up approach:

Fibonacci with less memory needed:

The Change Maker Problem

How do you make change adding up to 37 cents in the U.S. coin denominations?

What if the coins are worth 10, 7, 2, and 1, and you wanted to make 14 cents?

Greedy:

Fewest coins:

ALGORITHM CHANGEMAKER(D, amt)
//Input: $D[0..d-1]$ array of coin denominations
//Input: amt desired coin total value
//Output: the minimum number of coins in D to sum to amt
if $amt = 0$ **then**
return 0
$min \leftarrow \infty$
for $i \leftarrow 0..d-1$ **do**
if $amt \geq D[i]$ **then**
$x \leftarrow 1 + ChangeMaker(D, amt - D[i])$
if $x < min$ **then**
$min \leftarrow x$
return min

Draw the exhaustive search tree for `D = [1, 3, 5], amt = 8`.

How many recursive calls are made?

How many times does it compute each subprogram?

Augmented version to remember the answers to recursive subproblems: *a memory function*

ALGORITHM CHANGEMAKER($D, amt, sols$)
//Input: $D[0..d-1]$ array of coin denominations
//Input: amt desired coin total value
//Input: $sols[0..maxAmt]$ saved solutions, initialized to all -1
//Output: the minimum number of coins in D to sum to amt
if $amt = 0$ **then**
return 0
// did we already compute this amount's result?
if $sols[amt] \geq 0$ **then**
return $sols[amt]$
// we need to compute this amount's result
$min \leftarrow \infty$
for $i \leftarrow 0..d-1$ **do**
if $amt \geq D[i]$ **then**
$x \leftarrow 1 + ChangeMaker(D, amt - D[i])$
if $x < min$ **then**
$min \leftarrow x$
// save this result before returning in case we need it again
$sols[amt] \leftarrow min$
return min

Smaller recursive call tree:

How many non-trivial calls are needed here? What is the largest number for a given value of amt?

Binomial Coefficients

Binomial coefficients are the values $C(n, k)$ in the binomial formula:

$$(a + b)^n = C(n, 0)a^n + \cdots + C(n, k)a^{n-k}b^k + \cdots + C(0, n)b^n.$$

Computing for the first few values of n:

Recall *Pascal's Triangle*:

A dynamic programming algorithm to compute $C(n, k)$:

ALGORITHM BINOMIAL(n, k)
// Input: n, the power for $(a + b)^n$
// Input: k, the exponent of b in the desired term in the expanded polynomial
// $C[0..n, 0..k]$ is a dynamic programming array
// Output: the coefficient of the $a^{n-k}b^k$ term in the expanded polynomial
for $i \leftarrow 0..n$ **do**
  for $j \leftarrow 0..min(i, k)$ **do**
    if $j = 0$ **or** $j = i$ **then**
      $C[i, j] \leftarrow 1$
    else
      $C[i, j] \leftarrow C[i - 1, j - i] + C[i - 1, j]$
return $C[i, k]$

Basic operation:

Analysis:

The Knapsack Problem

Given n items with weights $w_1, w_2, ..., w_n$ and values $v_1, v_2, ..., v_n$, what is the most vaulable subset of items that can fit into a knapsack that has a total weight capacity of W.

What was the brute force approach we considered earlier? What is its Big-Θ efficiency?

Example instance, $W = 5$, weights and values as in the leftmost column of the table below.

Let's build a table V, where $V[i, j]$ is the optimal answer when we select from among only the first i items and limit to a capacity of j. Note that $V[4, 5]$ would then contain our solution.

		capacity j					
	i	0	1	2	3	4	5
	0						
$w_1 = 2$, $v_1 = 12$	1						
$w_2 = 1$, $v_2 = 10$	2						
$w_3 = 3$, $v_3 = 20$	3						
$w_4 = 2$, $v_4 = 15$	4						

Recurrence:

How about a top-down approach, where we use a memory function to compute only those subproblems we actually need?

		capacity j					
	i	0	1	2	3	4	5
	0						
$w_1 = 2, v_1 = 12$	1						
$w_2 = 1, v_2 = 10$	2						
$w_3 = 3, v_3 = 20$	3						
$w_4 = 2, v_4 = 15$	4						

Time efficiency: Space efficiency: