Dynamic Programming

CIS 315

"We now turn to the two sledgehammers of the algorithms craft, *dynamic programming* and *linear programming*, techniques of very broad applicability that can be invoked when more specialized methods fail. Predictably, this generality often comes with a cost in efficiency."

-- *Dasgupta, Papadimitriou, Vazirani*

comments:

- one point is that dynamic programming is not "elegant"
- we won't be covering linear programming here in 315

dynamic programming manifesto

CLRS, p 357:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from the computed information.

rod cutting example from text (p 360)

Given a rod of length n inches and a table of prices p_i for i=1,2,…,n (p_i is the price charged for a piece of length i inches), determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	30

with n=10

- two pieces of length 5: $p_5+p_5 = 20$
- no cuts: $p_{10}=30$
- six and four inches: $p_4+p_6 = 9+17 = 26$

crucial requirement

optimal substructure

For r_n, in an optimal solution, consider the first cut of length i. In that optimal solution, the remaining n-i inches must be cut optimally, obtaining revenue r_{n-i}.

first two steps of solution

subproblem:

r_i is the maximum revenue obtainable from a rod of i inches

(i=0,1,2,…,n)

recurrence:

look at the all possible first cuts:

$r_0 = 0$ (base case)

$r_n = \max_{1 \le i \le n}[p_i + r_{n-i}]$

fill out a table

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0										

using recurrence: $r_1 = MAX[p_1+r_0] = 1+0 = 1$

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0	1									

$r_2 = MAX[p_1+r_1, p_2+r_0] = MAX[1+1, 5+0] = 5$

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0	1	5								

$r_3 = MAX[p_1+r_2, p_2+r_1 , p_3+r_0] = MAX[1+5, 5+1, 8+0] = 8$
$r_4 = MAX[p_1+r_3, p_2+r_2 , p_3+r_1 , p_4+r_0] = MAX[1+8, 5+5, 8+1, 9+0] = 10$

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0	1	5	8	10						

$r_5 = MAX[p_1+r_4, p_2+r_3 , p_3+r_2 , p_4+r_1 , p_5+r_0] = MAX[1+10, 5+8, 8+5, 9+1, 10+0] = 13$
$r_6 = MAX[p_1+r_5, p_2+r_4 , p_3+r_3 , p_4+r_2 , p_5+r_1, p_6+r_0] =$
$MAX[1+13, 5+10, 8+8, 9+5, 10+1, 17+0] = 17$

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0	1	5	8	10	13	17				

clearly an $O(n^2)$ process this way

bottom-up version

```
array r[0…n] of int

r[0] = 0

for j = 1 to n
      q = -∞
      for i = 1 to j
            q = max[ q, p[i]+r[j-i] ]
     r[j] = q

return r[n]
```

versus naïve recursive version

```
CutRod(p,n)

  if n=0 return 0

  q = -∞

  for i = 1 to n
      q = MAX[ q, p[i]+CutRod(p, n-i) ]

  return q
```

$O(2^n)$ time according to text

so memoize it