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.
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.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

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 all possible first cuts:
  - \( r_0 = 0 \) (base case)
  - \( r_n = \max_{1 \leq i \leq n}[p_i + r_{n-i}] \)
fill out a table

<table>
<thead>
<tr>
<th></th>
<th>r₀</th>
<th>r₁</th>
<th>r₂</th>
<th>r₃</th>
<th>r₄</th>
<th>r₅</th>
<th>r₆</th>
<th>r₇</th>
<th>r₈</th>
<th>r₉</th>
<th>r₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

using recurrence: \( r₁ = \text{MAX}[p₁+r₀] = 1+0 = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>r₀</th>
<th>r₁</th>
<th>r₂</th>
<th>r₃</th>
<th>r₄</th>
<th>r₅</th>
<th>r₆</th>
<th>r₇</th>
<th>r₈</th>
<th>r₉</th>
<th>r₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( r₂ = \text{MAX}[p₁+r₁, p₂+r₀] = \text{MAX}[1+1, 5+0] = 5 \)

<table>
<thead>
<tr>
<th></th>
<th>r₀</th>
<th>r₁</th>
<th>r₂</th>
<th>r₃</th>
<th>r₄</th>
<th>r₅</th>
<th>r₆</th>
<th>r₇</th>
<th>r₈</th>
<th>r₉</th>
<th>r₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( r_3 = \text{MAX}[p_1 + r_2, p_2 + r_1, p_3 + r_0] = \text{MAX}[1+5, 5+1, 8+0] = 8 \)
\( r_4 = \text{MAX}[p_1 + r_3, p_2 + r_2, p_3 + r_1, p_4 + r_0] = \text{MAX}[1+8, 5+5, 8+1, 9+0] = 10 \)

\[\begin{array}{cccccccccc}
 r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 & r_{10} \\
 0 & 1 & 5 & 8 & 10 & & & & & & \\
\end{array}\]

\( r_5 = \text{MAX}[p_1 + r_4, p_2 + r_3, p_3 + r_2, p_4 + r_1, p_5 + r_0] = \text{MAX}[1+10, 5+8, 8+5, 9+1, 10+0] = 13 \)
\( r_6 = \text{MAX}[p_1 + r_5, p_2 + r_4, p_3 + r_3, p_4 + r_2, p_5 + r_1, p_6 + r_0] = \text{MAX}[1+13, 5+10, 8+8, 9+5, 10+1, 17+0] = 17 \)

\[\begin{array}{cccccccccc}
 r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 & r_{10} \\
 0 & 1 & 5 & 8 & 10 & 13 & 17 & & & & \\
\end{array}\]

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 = -\infty
    for i = 1 to j
        q = \text{max}[ q, p[i]+r[j-i] ]
    r[j] = q

return r[n]
versus naïve recursive version

\[
\text{CutRod}(p, n)
\]

\[
\begin{align*}
\text{if } n &= 0 \quad \text{return } 0 \\
q &= -\infty \\
\text{for } i &= 1 \text{ to } n \\
q &= \max[ q, p[i] + \text{CutRod}(p, n-i) ] \\
\text{return } q
\end{align*}
\]

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

so memoize it