Notes, 7(~~a)~~ (b)

ECE 606

Dynamic Programming, Examples

Interval Scheduling

If one is unable to see that Interval Scheduling possesses greedy choice, all is not lost. As your textbook documents (from CLRS), it possesses optimal substructure which can then be exploited by dynamic programming. Your textbook (CLRS) calls it "an activity-selection problem."

Rod Cutting

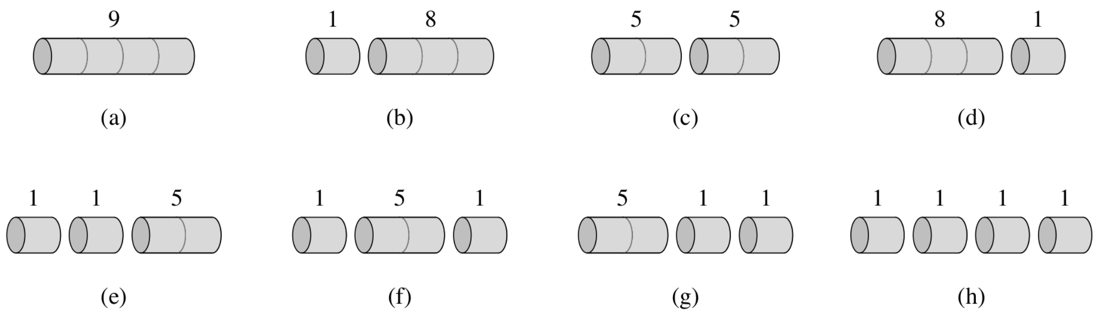
In this problem, we are given a rod of n inches. We are given also a price-list which tells us for what price we can sell a rod of i inches for $i = 1, 2, \ldots$. The question is: in what way should ~~b~~we cut our rod of n inches to maximize our revenue?

Example of a price-list from your textbook:

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

For example, if $n = 4$, the best choice is to cut make one cut of the rod into two equal pieces each of length 2. Then, our revenue is 10.

Naive algorithm: brute-force check for every inch of the rod, whether it is good to cut at that inch. Number of possibilities: 2^{n-1}.



Optimal substructure & recurrence for optimum

We can always think of cutting from one end to the other, e.g., right to left, only. Suppose our rod is length x. Why would be place a cut at a particular length $y \in \{1, 2, \ldots, \lfloor x/2 \rfloor\}$? If $R[x]$ is the maximum revenue we can achieve from a rod of length x, then we seek:

$$y \text{ that achieves } \max_{y \in \{1, \ldots, \lfloor x/2 \rfloor\}} \{p_x, p_y + R[x - y]\}$$

$$R[x] = \begin{cases} -\infty & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ \max\limits_{y \in \{1, \ldots, \lfloor x/2 \rfloor\}} \{p_x, p_y + R[x-y]\} & \text{otherwise} \end{cases}$$

```
RodCut-DP($p, n$)
1 $R \leftarrow$ new array $[0, \ldots, n]$
2 $R[0] \leftarrow 0$, $R[1] \leftarrow p[1]$
3 foreach $x$ from 2 to $n$ do
4     $R[x] \leftarrow p[x]$
5     foreach $y$ from 1 to $\lfloor x/2 \rfloor$ do
6         if $R[x] < p[y] + R[x-y]$ then $R[x] \leftarrow p[y] + R[x-y]$
7 return $R[n]$
```

But that gives us only the max revenue that can be achieved. We also want to record exactly which cuts to make. One way to do this: for each x, compute $C[x]$, which is the first cut that we make to a rod of length x. If this cut is at $y = 0$, we are done. Otherwise, we then consult $C[x - y]$ for the next cut.

For example, if $C[10] = 2$, then first cut at 2 inches, and then consult $C[8]$. If $C[10] = 0$, then we make no cuts to the rod of 10 inches.

```
RodCut-DP-with-Cuts($p, n$)
1  $R \leftarrow$ new array $[0, \ldots, n]$, $C \leftarrow$ new array $[0, \ldots, n]$
2  $R[0] \leftarrow 0$, $R[1] \leftarrow p[1]$
3  $C[0] \leftarrow 0$, $C[1] \leftarrow 0$
4  foreach $x$ from 2 to $n$ do
5      $R[x] \leftarrow p[x]$, $C[x] \leftarrow 0$
6      foreach $y$ from 1 to $\lfloor x/2 \rfloor$ do
7          if $R[x] < p[y] + R[x - y]$ then
8              $R[x] \leftarrow p[y] + R[x - y]$
9              $C[x] \leftarrow y$
10 return $\langle R[n], C \rangle$
```

Matrix chain multiplication

This is the problem of computing $A_1 A_2 \ldots A_n$, where each A_i is a matrix. Of course, the product exists if and only if given that A_i is $n_i \times m_i$ and A_{i+1} is $n_{i+1} \times m_{i+1}$, $m_i = n_{i+1}$.

You may recall that given X that is $n \times m$ and Y that is $m \times p$, the product $Z = XY$ is $n \times p$, where $z_{i,j} = \sum_{k=1}^{m} x_{i,k} \cdot y_{k,j}$.

One way to characterize the cost of computing $Z = XY$ is to count the number of scalar multiplications. We have m scalar multiplications to compute each entry in Z. Thus, the total $= nmp$.

Now consider we want to compute $A_1 A_2 A_3$. Because of associativity, we can compute this either as: $(A_1 A_2) A_3$ or $A_1 (A_2 A_3)$. Does it make a difference? Not to correctness, but may to efficiency as measured by the number of scalar multiplications.

E.g., if A_1 is 1×10, A_2 is 10×1 and A_3 is 1×10, then:

- $(A_1 A_2) A_3$ requires $1 \times 10 \times 1 + 1 \times 1 \times 10 = 20$ scalar multiplications.
- $A_1 (A_2 A_3)$ requires $10 \times 1 \times 10 + 1 \times 10 \times 10 = 200$ scalar multiplications.

So: a difference of an order of magnitude.

In matrix chain multiplication, we are given the dimensions $p_{i-1} \times p_i$ of n matrices. We are asked to identify the parenthesizations, which in turn specify the sequence of matrix multiplications we will perform. Suppose $m[i,j]$ is the minimum cost, i.e., # scalar multiplications, to compute $A_i A_{i+1} \ldots A_j$. Then:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{otherwise} \end{cases}$$

The above recurrence can be realized bottom-up, using dynamic programming.