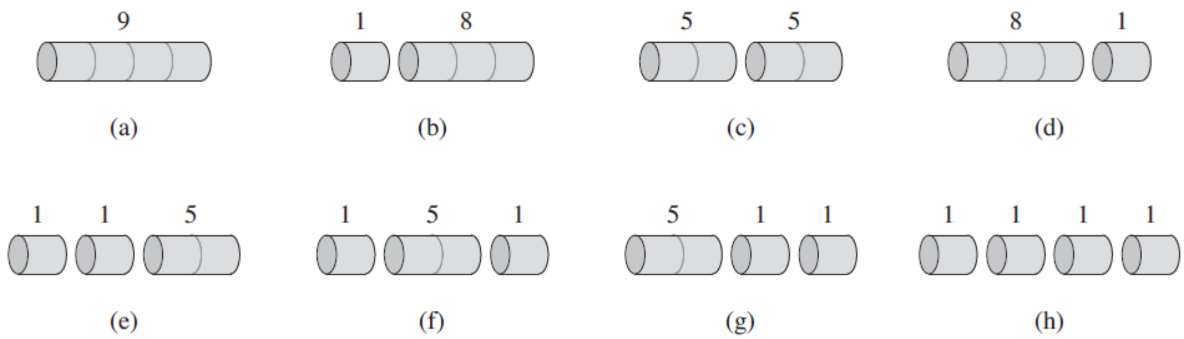
Dynamic Programming (14 points)

The rod-cutting problem is the following. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtain- able by cutting up the rod and selling the pieces. Note that if the price p_n for a rod of length n is large enough, an optimal solution may require no cutting at all.

Consider the case when $n = 4$. The figure below shows all the ways to cut up a rod of 4 inches in length, including the way with no cuts at all. We see that cutting a 4-inch rod into two 2-inch pieces produces revenue $p_2 + p_2 = 5 + 5 = 10$, which is optimal.



1. [2 points] Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the density of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

   Solution: Let $p_1 = 0, p_2 = 4, p_3 = 7$ and $n = 4$. The greedy strategy would first cut off a piece of length 3 since it has highest density. The remaining rod has length 1, so the total price would be 7. On the other hand, two rods of length 2 yield a price of 8.

2. [3 points] Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

   Solution: Now, instead of equation (15.1), we have that

   $$r_n = \max\{p_n, r_1 + r_{n-1} - c, r_2 + r_{n-2} - c, \ldots, r_{n-1} + r_1 - c\}$$

   And so, to change the top down solution to this problem, we would change MEMOIZED-CUT-ROD-AUX(p, n, r) as follows. The upper bound for i on line 6 should be $n-1$ instead of n . Also, after the for loop, but before line 8, set $q = \max\{q - c, p[i]\}$

3. [3 points] Modify MEMOIZED-CUT-ROD to return not only the value but the actual solution, too.

Solution: Create a new array called s . Initialize it to all zeros in MEMOIZED-CUT-ROD(p, n) and pass it as an additional argument to MEMOIZED-CUT-ROD-AUX(p, n, r, s). Replace line 7 in MEMOIZED-CUT-ROD-AUX by the following: $t = p[i]+$MEMOIZED-CUT-ROD-AUX$(p, n-i, r, s)$. Following this, if $t > q$, set $q = t$ and $s[n] = i$. Upon termination, $s[i]$ will contain the size of the first cut for a rod of size i.

4. [2 points] Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6\rangle$. What is the total cost?

Solution: An optimal parenthesization of that sequence would be $(A_1A_2)\,((A_3A_4)\,(A_5A_6))$ which will require $5{*}50{*}6+3{*}12{*}5+5{*}10{*}3+3{*}5{*}6+5{*}3{*}6 = 1500+180+150+90+90 = 2010$

5. [4 points] Give a recursive algorithm *Matrix-Chain-Multiply*(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, \ldots, A_n\rangle$, the s table computed by *Matrix-Chain-Order*, and the indices i and j. (The initial call would be *Matrix-Chain-Multiply*$(A, s, 1, n)$.

Solution: The following algorithm actually performs the optimal multiplication, and is recursive in nature:

Matrix-Chain-Multiply(A, s, i, j)
1. if $i == j$
2. return A_i
3. return *Matrix-Chain-Multiply*$(A, s, i, s[i, j])$ $\times$ *Matrix-Chain-Multiply*$(A, s, s[i, j]+1, j)$