CSCI 470: Dynamic Programming II: Rod Cutting Problem, Longest Common Subsequence

Vijay Chaudhary

November 15, 2023

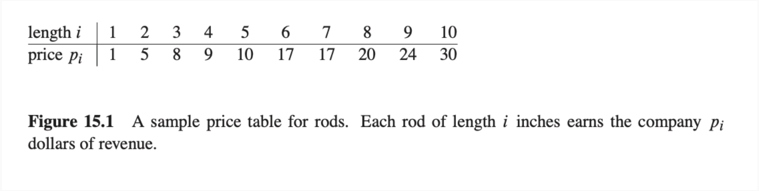
Department of Electrical Engineering and Computer Science
Howard University

Overview

Dynamic Programming: Rod Cutting Problem

Rod Cutting Problem

- We assume that we know, for $i = 1, 2, \ldots$, the price p_i in dollars that Serling Enterprises charges for a rod of length i inches. Rod lengths are always an integral number of inches. Figure 15.1 gives a sample price table.



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

Figure 15.1 A sample price table for rods. Each rod of length i inches earns the company p_i dollars of revenue.

- Given a rod of length n inches, a table of prices p_i, for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtainable 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.

Rod Cutting Problem

- Consider the case when $n = 4$. Figure 15.2 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.
- We can cut up a rod of length n in 2^{n-1} different ways, since we have an independent option of cutting, or not cutting, at distance i inches from the left end, for $i = 1, 2, ..., n - 1$.

Rod Cutting Problem

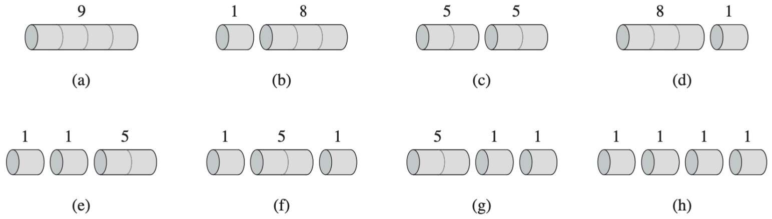


Figure 15.2 The 8 possible ways of cutting up a rod of length 4. Above each piece is the value of that piece, according to the sample price chart of Figure 15.1. The optimal strategy is part (c)—cutting the rod into two pieces of length 2—which has total value 10.

Solutions to our sample problem

For our sample problem, we can determine the optimal revenue figures r_i, for $i = 1, 2, ..., 10$, by inspection, with the corresponding optimal decompositions

$r_1 = 1$ from solution $1 = 1$ (no cuts),

$r_2 = 5$ from solution $2 = 2$ (no cuts),

$r_3 = 8$ from solution $3 = 3$ (no cuts),

$r_4 = 10$ from solution $4 = 2 + 2$,

$r_5 = 13$ from solution $5 = 2 + 3$,

$r_6 = 17$ from solution $6 = 6$ (no cuts),

$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,

$r_8 = 22$ from solution $8 = 2 + 6$,

$r_9 = 25$ from solution $9 = 3 + 6$,

$r_10 = 30$ from solution $10 = 10$ (no cuts).

Optimal Solution

More generally, we can frame the values r_n for $n \geq 1$ in terms of optimal revenues from shorter rods:

$$r_n = \max(p_n, r_1 + r_n, r_2 + r_{n-2}, ..., r_{n-1} + r_1). \tag{1}$$

Reducing to an Optimal Solution

- In a related, but slightly simpler, way to arrange a recursive structure for the rod-cutting problem, we view a decomposition as consisting of a first piece of lenght i cut off the left-hand end, and then a right-hand remainder of length $n - i$.
- Only the remainder, and not the first piece, may be further divided. We may view every decomposition of a length-n rod in this way: as a first piece followed by some decomposition of the remainder.
- When doing so, we can couch the solution with no cuts at all as saying the first piece has size $i = n$ and revenue p_n and that the remainder has size 0 with corresponding revenue $r_0 = 0$.

Reducing ...

We thus obtain the following simpler version of equation (1):

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}). \tag{2}$$

In this formulation, an optimal solution embodies the solution to only *one* related subproblem - the remainder - rather than two.

Top Down Approach

```
CUT-ROD($p, n$)
1  if $n == 0$
2      return 0
3  $q = -\infty$
4  for $i = 1$ to $n$
5      $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$
6  return $q$
```

Inefficiency?

- Why is Cut-Rod so inefficient?

Inefficiency?

- Why is Cut-Rod so inefficient?
- The problem is that Cut-Rod(p, n) calls itself recursively over and over again with the same parameter values; it solves the same subproblems repeatedly.

Inefficiency with a recursion tree

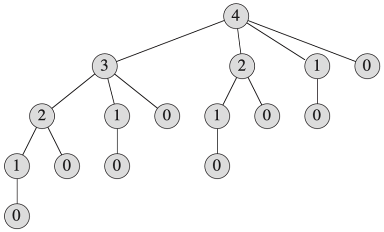


Figure 15.3 The recursion tree showing recursive calls resulting from a call CUT-ROD(p, n) for $n = 4$. Each node label gives the size n of the corresponding subproblem, so that an edge from a parent with label s to a child with label t corresponds to cutting off an initial piece of size $s - t$ and leaving a remaining subproblem of size t. A path from the root to a leaf corresponds to one of the 2^{n-1} ways of cutting up a rod of length n. In general, this recursion tree has 2^n nodes and 2^{n-1} leaves.

Runtime

To analyze the running time of CUT-ROD, let $T(n)$ denote the total number of calls made to CUT-ROD when called with its second parameter equal to n. This expression equals the number of nodes in a subtree whose root is labeled n in the recursion tree. The count includes the initial call at its root. Thus, $T(0) = 1$ and

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j). \tag{3}$$

The initial 1 is for the call at the root, and the term $T(j)$ counts the number of calls (including recursive calls) due to the call CUT-ROD$(p, n-i)$, where $j = n - i$.

DP with Top Down Approach

```
MEMOIZED-CUT-ROD($p, n$)
1  let $r[0..n]$ be a new array
2  for $i = 0$ to n
3      $r[i] = -\infty$
4  return MEMOIZED-CUT-ROD-AUX($p, n, r$)
```

DP Top Down

```
MEMOIZED-CUT-ROD-AUX($p, n, r$)
1  if $r[n] \geq 0$
2      return $r[n]$
3  if $n$ == 0
4      $q = 0$
5  else
6      $q = -\infty$
7      for $i = 1$ to $n$
8          $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n-i, r))$
9  $r[n] = q$
10 return $q$
```

Bottom Up

```
BOTTOM-UP-CUT-ROD($p, n$)
1  let $r[0..n]$ be a new array
2  $r[0] = 0$
3  for $j = 1$ to n
4      $q = -\infty$
5      for $i = 1$ to $j$
6          $q = \max(q, p[i] + r[j-i])$
7      $r[j] = q$
8  return $r[n]$
```

Runtime

- What is the runtime of the procedures above?

Runtime

- What is the runtime of the procedures above?
- Top-Down?

Runtime

- What is the runtime of the procedures above?
- Top-Down?
- Bottom-Up?

Subproblem Graph

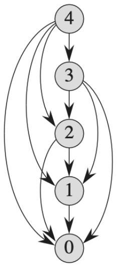


Figure 15.4 The subproblem graph for the rod-cutting problem with $n = 4$. The vertex labels give the sizes of the corresponding subproblems. A directed edge (x, y) indicates that we need a solution to subproblem y when solving subproblem x. This graph is a reduced version of the tree of Figure 15.3, in which all nodes with the same label are collapsed into a single vertex and all edges go from parent to child.

Subproblem Graph

- The bottom-up method for dynamic programming considers the vertices of the subproblem graph in such an order that we solve the subproblems y adjacent to a given subproblem x before we solve subproblem x.

Subproblem Graph

- The bottom-up method for dynamic programming considers the vertices of the subproblem graph in such an order that we solve the subproblems y adjacent to a given subproblem x before we solve subproblem x.
- In a bottom-up dynamic-programming algorithm, we consider the vertices of the subproblem graph in an order that is a "reverse topological sort," or a "topological sort of the transpose" of the subproblem graph.

Subproblem Graph

- The bottom-up method for dynamic programming considers the vertices of the subproblem graph in such an order that we solve the subproblems y adjacent to a given subproblem x before we solve subproblem x.
- In a bottom-up dynamic-programming algorithm, we consider the vertices of the subproblem graph in an order that is a “reverse topological sort,” or a “topological sort of the transpose” of the subproblem graph.
- In other words, no subproblem is considered until all of the subproblems it depends upon have been solved.

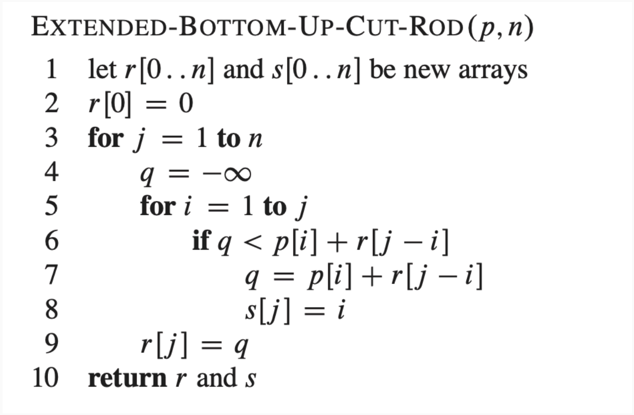
Subproblem Graph

- The bottom-up method for dynamic programming considers the vertices of the subproblem graph in such an order that we solve the subproblems y adjacent to a given subproblem x before we solve subproblem x.
- In a bottom-up dynamic-programming algorithm, we consider the vertices of the subproblem graph in an order that is a "reverse topological sort," or a "topological sort of the transpose" of the subproblem graph.
- In other words, no subproblem is considered until all of the subproblems it depends upon have been solved.
- Similarly, using notions from the same chapter, we can view the top-down method (with memoization) for dynamic programming as a "depth-first search" of the subproblem graph.

Runtime with Subproblem Graph

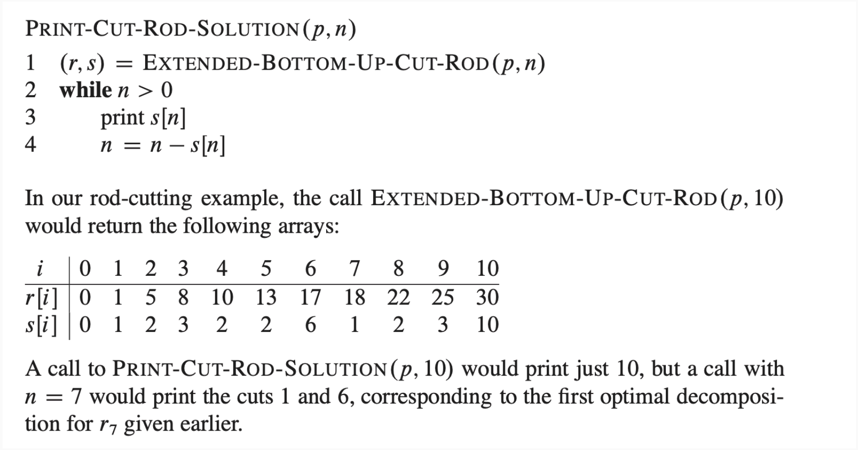
- The size of the subproblem graph $G = (V, E)$ can help us determine the running time of the dynamic programming algorithm.
- Since we solve each subproblem just once, the running time is the sum of the times needed to solve each subproblem. Typically, the time to compute the solution to a subproblem is proportional to the degree (number of outgoing edges) of the corresponding vertex in the subproblem graph, and the number of subproblems is equal to the number of vertices in the subproblem graph.

Printing Optimal Solution



```
EXTENDED-BOTTOM-UP-CUT-ROD$(p, n)$
let $r[0..n]$ and $s[0..n]$ be new arrays
$r[0] = 0$
for $j = 1$ to $n$
    $q = -\infty$
    for $i = 1$ to $j$
        if $q < p[i] + r[j-i]$
            $q = p[i] + r[j-i]$
            $s[j] = i$
    $r[j] = q$
return $r$ and $s$
```

Printing Optimal Solution



```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n − s[n]
```

In our rod-cutting example, the call EXTENDED-BOTTOM-UP-CUT-ROD$(p, 10)$ would return the following arrays:

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

A call to PRINT-CUT-ROD-SOLUTION$(p, 10)$ would print just 10, but a call with $n = 7$ would print the cuts 1 and 6, corresponding to the first optimal decomposition for r_7 given earlier.

LCS

LCS

- Formally, given a sequence $X = \langle x_1, x_2, ..., x_m \rangle$, another sequence $Z = \langle z_1, z_2, ..., z_k \rangle$ is a subsequence of X if there exists a strictly increasing sequence $\langle i_1, i_2, ..., i_k \rangle$ of indices of X such that for all $j = 1, 2, ..., k$ we have $x_{ij} = z_j$.
- For example, $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ with corresponding index sequence $\langle 2, 3, 5, 7 \rangle$