ADVANCED ANALYSIS OF ALGORITHMS
CPS 5440

UNIT 11: DYNAMIC PROGRAMMING. ROD CUTTING.

ROD CUTTING

- Serling Enterprises buys long steel rods and cuts them into shorter rods, which it then sells.
- Each cut is free.
- The management of Serling Enterprises wants to know the best way to cut up the rods.

ROD CUTTING

- 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

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

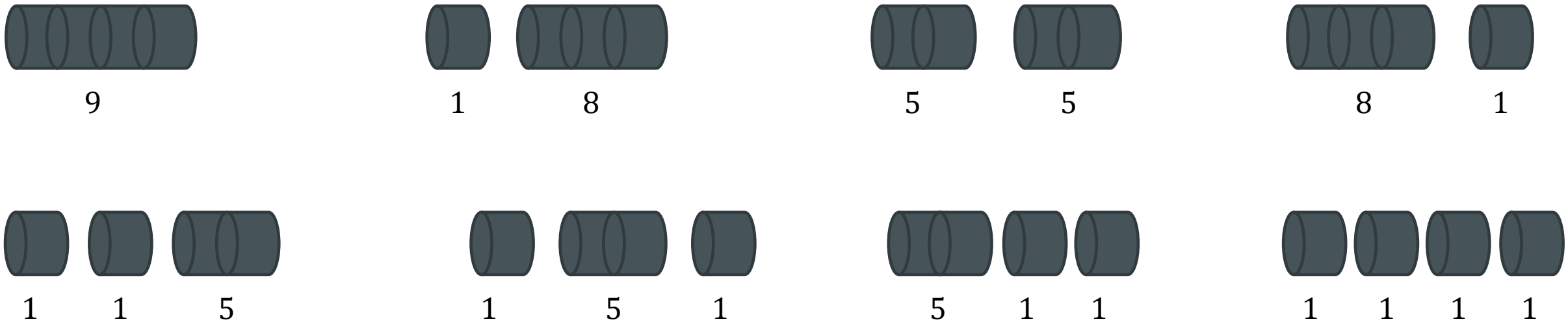
A sample price table for rods. Each rod of length i inches earns the company p_i dollars of revenue.

ROD CUTTING

- 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 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

8 possible ways of cutting up a rod of length 4.
Above each piece is the value of that piece, according to the price chart.
The optimal strategy is cutting the rod into two pieces of length 2, which has a total value of 10.



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

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

ROD CUTTING

- 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 $i = 1,2, \ldots, n-1$,

ROD CUTTING

- We denote a decomposition into pieces using ordinary additive notation, so that $7 = 2 + 2 + 3$ indicates that a rod of length 7 is cut into three pieces, two of length 2 and one of length 3.
- If an optimal solution cuts the rod into k pieces, for some $1 \leq k \leq n$, then an optimal decomposition $n = i_1 + i_2 + \ldots + i_k$ of the rod into pieces of lengths $i_1, i_2, \ldots, i_k$ provides maximum corresponding revenue $r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$

ROD CUTTING

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

$r_1 = 1$ 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 solution $7 = 2 + 2 + 3$;
$r_8 = 22$ from solution $8 = 2 + 6$;
$r_9 = 25$ from solution $9 = 3 + 6$;
$r_{10} = 30$ from solution $10 = 10$ (no cuts) :

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

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

ROD CUTTING

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

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

- p_n: corresponds to making no cuts at all and selling the rod of length n as is
- The other $n - 1$ arguments to max correspond to the maximum revenue obtained by making an initial cut of the rod into two pieces of size i and $n - i$, for each $i = 1, 2, \dots, n - 1$, and then optimally cutting up those pieces further, obtaining revenues r_i and r_{n-i} from those two pieces

ROD CUTTING

- Note that to solve the original problem of size n, we solve smaller problems of the same type, but of smaller sizes.
- Once we make the first cut, we may consider the two pieces as independent instances of the rod-cutting problem.
- The overall optimal solution incorporates optimal solutions to the two related subproblems, maximizing revenue from each of those two pieces.
- We say that the rod-cutting problem exhibits optimal substructure: optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently

ROD CUTTING

- Simple recursive structure:
- A decomposition can consist of a first piece of length 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 length$-n$ as a first piece followed by some decomposition of the remainder

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

ROD CUTTING

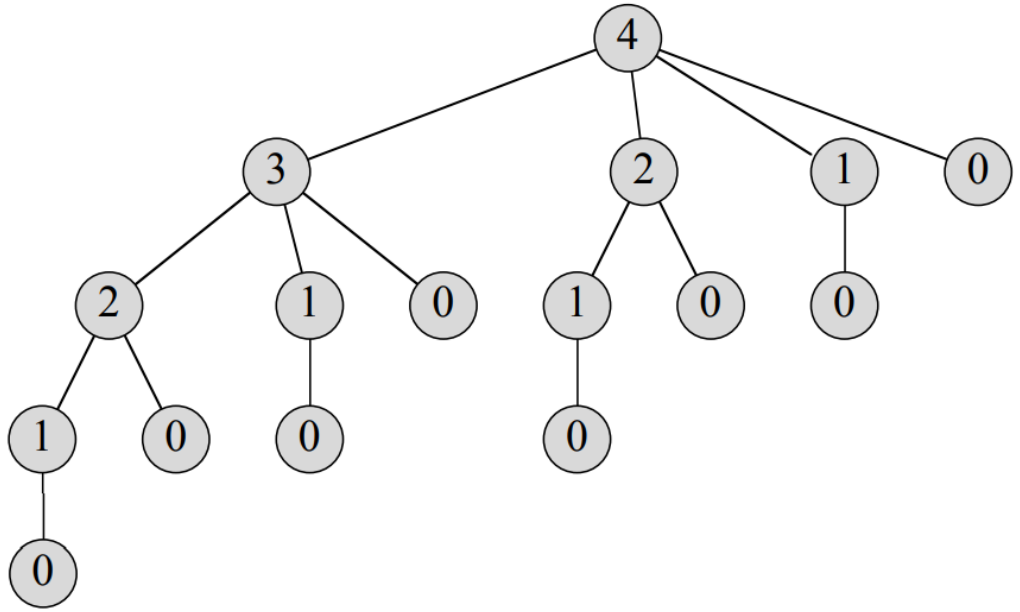
- Recursive version
- Input: an array $p[1..n]$ of prices and an integer n
- Output: the maximum revenue possible for a rod of length n

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

Cut-ROD so inefficient !

ROD CUTTING

- CUT-ROD calls itself recursively over and over again with the same parameter values; it solves the same subproblems repeatedly



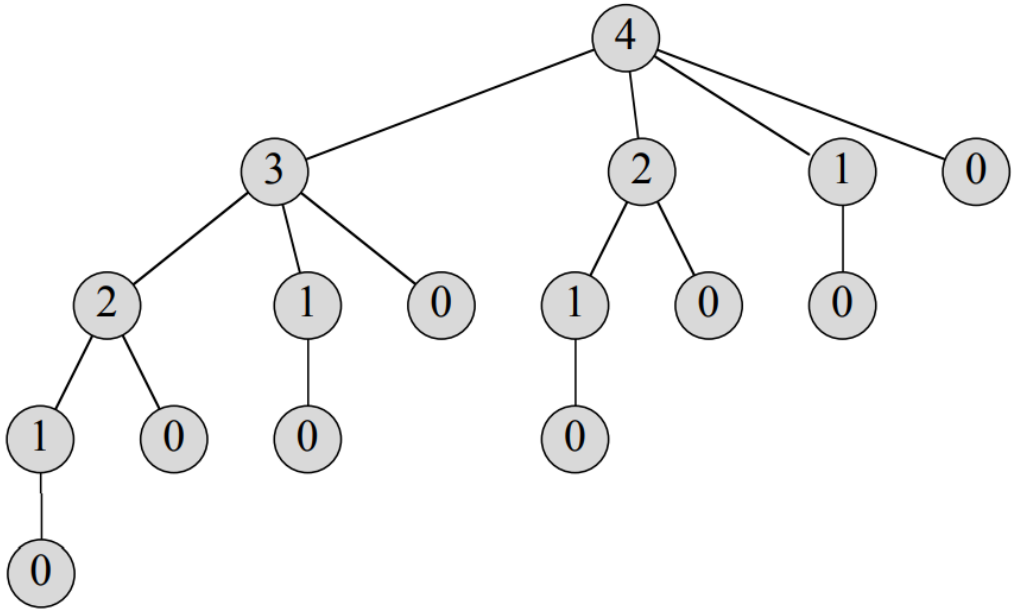
The recursion tree showing recursive calls resulting from a call $CUT - ROD(p, n)$ for $n = 4$.
Each node label gives the size n of the corresponding subproblem, so that an edge from a parent with label s to a child with label t corresponds to cutting off an initial piece of size $s - t$ and leaving 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

ROD CUTTING

- 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.

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

- 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 $j = n - i$



- $T(n) = 2^n$

ROD CUTTING

- In retrospect, this exponential running time is not so surprising. CUT-ROD explicitly considers all the 2^{n-1} possible ways of cutting up a rod of length n.
- The tree of recursive calls has 2^{n-1} leaves, one for each possible way of cutting up the rod.
- The labels on the simple path from the root to a leaf give the sizes of each remaining right-hand piece before making each cut.
- That is, the labels give the corresponding cut points, measured from the right-hand end of

ROD CUTTING

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

Top-down with memoization

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

Bottom-up DP

- This approach typically depends on some natural notion of the "size" of a subproblem, such that solving any particular subproblem depends only on solving "smaller" subproblems
- We sort the subproblems by size and solve them in size order, smallest first.
- When solving a particular subproblem, we have already solved all of the smaller subproblems its solution depends upon, and we have saved their solutions
- We solve each subproblem only once, and when we first see it, we have already solved all of its prerequisite subproblems.
- The bottom-up approach often has much better constant factors, since it has less overhead for procedure calls.

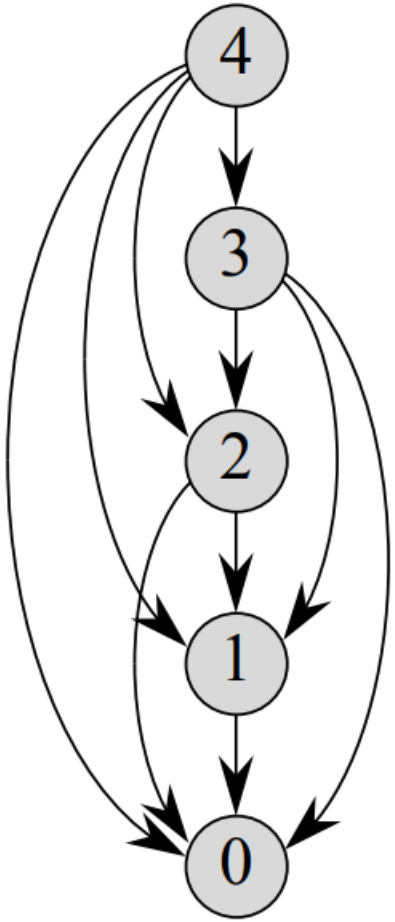
Bottom-up DP

```
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 = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

- A subproblem of size i is "smaller" than a subproblem of size j if $i < j$
- Thus, the procedure solves subproblems of size $j = 0,1, \ldots, n$, in that order

Bottom-up DP

```
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 = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```



- The subproblem graph for the rod-cutting problem with $n = 4$.
  - A directed edge (x, y) indicates that we need a solution to subproblem y when solving subproblem x.

DP Time Complexity

- The bottom-up and top-down versions have the same asymptotic running time.
- The running time of procedure BOTTOM-UP-CUT-ROD is $\theta(n^2)$due to its doubly-nested loop structure
- The top-down approach solves each subproblem just once. It solves subproblems for sizes $0,1, \ldots, n$.

Reconstructing a solution

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
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]
```

Reconstructing a solution

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

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 its optimal decomposition

QUESTIONS/ANSWERS