Lecture 18

Dynamic Programming: Rod Cutting to maximize profit, Problem Analysis

Dynamic Programming

Dynamic Programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub instances

- *Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS*

- *"Programming" here means "planning"*
- ***Main idea**:*
  - *set up a recurrence relating a solution to a larger instance to solutions of some smaller instances*
  - *solve smaller instances once*
  - *record solutions in a table*
  - *extract solution to the initial instance from that table*

Dynamic programming

- ***Dynamic programming*** *is typically applied to optimization problems. In such problem there can be* ***many solutions****. Each solution has a value, and we wish to find a* ***solution*** *with the optimal value.*

The development of a dynamic programming

1. Characterize the structure of an optimal solution.

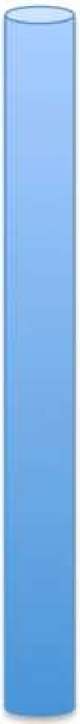
2. Recursively define the value of an optimal solution.

3. Compute the value of an optimal solution in a bottom-up fashion.

4. Construct an optimal solution from computed information.

Rod cutting: *To maximize the profit*

- ***Input:*** *A length n and table of prices p_i, for $i = 1, 2, \ldots, n$.*
- ***Output:*** *The maximum revenue obtainable for rods whose lengths sum to n, computed as the sum of the prices for the individual rods.*



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

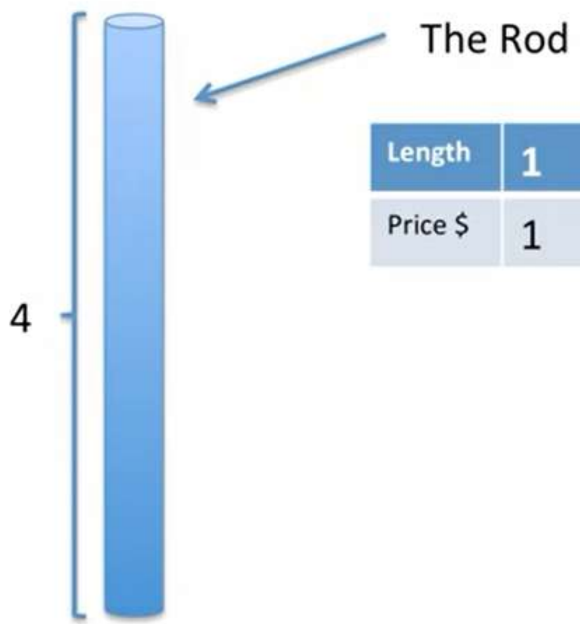
Recursive Solution

```
RodCutting(length, Price[ ])
    if(length == 0)
        return 0
    max = −∞
    for(i = 1; i ≤ length; i++)
        tmp = Price[i] + RodCutting(length−i, Price)
        if(tmp > max)
            max = tmp
    return max
```

Rod Cutting Example

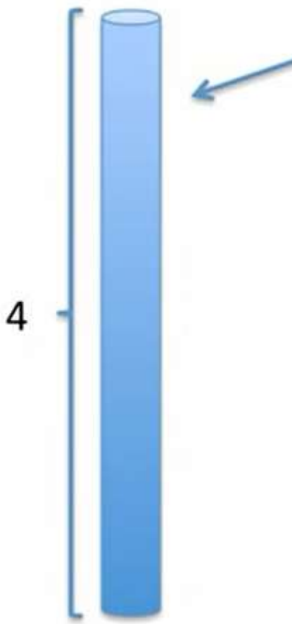
Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Rod Cutting Example



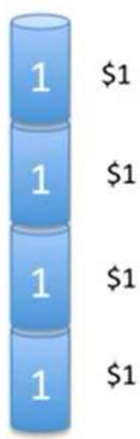
Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Rod Cutting Example



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Possible Combinations:

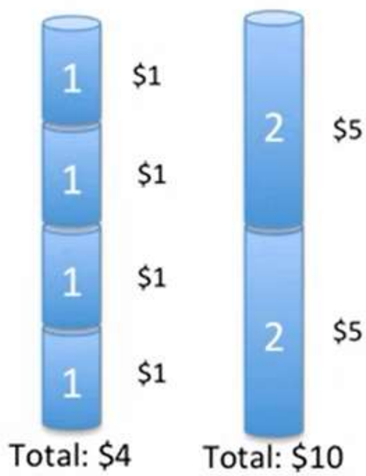


Total: $4

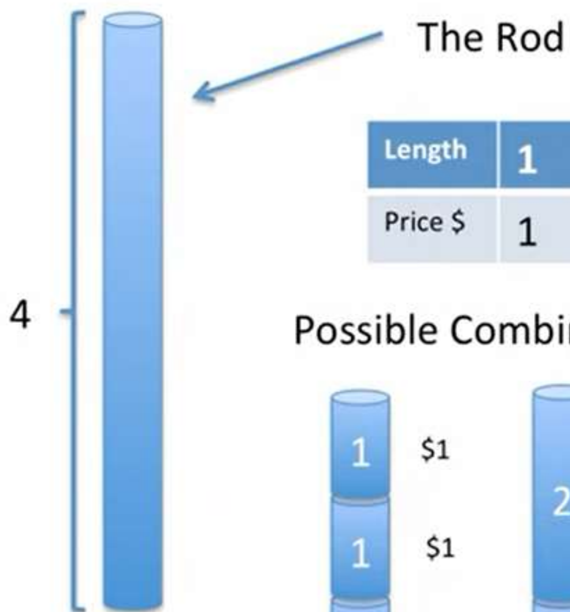
Rod Cutting Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Possible Combinations:



Rod Cutting Example



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

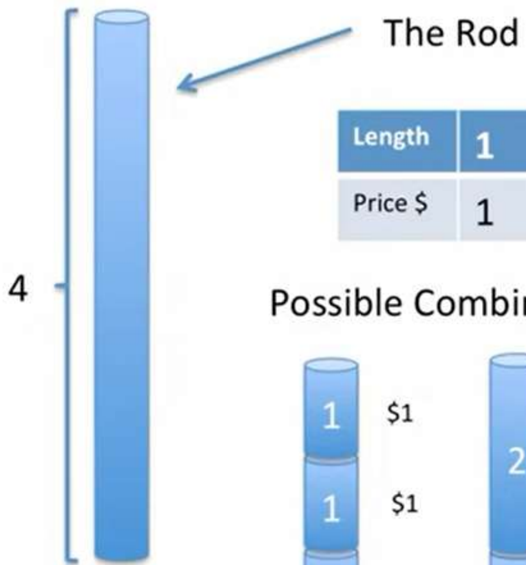
Possible Combinations:

Rod Cutting Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Possible Combinations:

Rod Cutting Example



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

Possible Combinations:

The remaining 3 ways to cut are just permutations of the left arrangements.

Rod Cutting Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

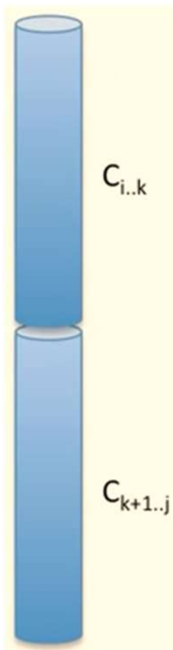
Possible Combinations:

The remaining 3 ways to cut are just permutations of the left arrangements.

Rod Cutting Dynamic Programming

Let's say we had the optimal solution for cutting the rod $C_{i...j}$ where C_i is the first piece, and C_j is the last piece.

If we take one of the cuts from this solution, somewhere in the middle, say k, and split it so we have two sub problems, $C_{i..k}$, and $C_{k+1..j}$ (Assuming our optimal is not just a single piece)



Let's assume we had a more optimal way of cutting $C_{i..k}$

We would swap the old $C_{i..k}$, and replace it with the more optimal $C_{i..k}$

Overall, the entire problem would now have an even more optimal solution!

But we already had stated that we had the optimal solution! This is a contradiction!

Therefore our original optimal solution is the optimal solution, and this problem exhibits optimal substructure.

Rod Cutting Solution

Let's define C(i) as the price of the optimal cut of a rod up until length i

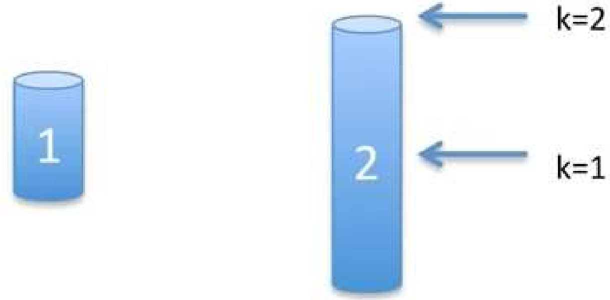
Let V_k be the price of a cut at length k

How to develop a solution:

We define the smallest problems first, and store their solutions.

We increase the rod length, and try all the cuts for that size of rod, taking the most profitable one.

We store the optimal solution for this sized piece, and build solutions to larger pieces from them in some sort of data structure.



$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Memoization

Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt								

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt								

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(1) = 1$$

Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1							

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(2) = \max \begin{cases} V_1 + C(1) = 1 + 1 = 2 \\ V_2 = 5 \end{cases}$$

Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5						

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(3) = \max \begin{cases} V_1 + C(2) = 1 + 5 = 6 \\ V_2 + C(1) = 5 + 1 = 6 \\ V_3 = 8 \end{cases}$$

Example

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8					

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

$$C(4) = \max \begin{cases} V_1 + C(3) = 1 + 8 = 9 \\ V_2 + C(2) = 5 + 5 = 10 \\ V_3 + C(1) = 8 + 1 = 9 \\ V_4 = 9 \end{cases}$$

Example

Rod Cutting DP

$$C(i) = \max_{1 \leq k \leq i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10				

$$C(5) = \max \begin{cases} V_1 + C(4) = 1 + 10 = 11 \\ V_2 + C(3) = 5 + 8 = 13 \\ V_3 + C(2) = 8 + 5 = 13 \\ V_4 + C(1) = 9 + 1 = 10 \\ V_5 = 10 \end{cases}$$

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13			

$$C(6) = \max \begin{cases} V_1 + C(5) = 1 + 13 = 14 \\ V_2 + C(4) = 5 + 10 = 15 \\ V_3 + C(3) = 8 + 8 = 16 \\ V_4 + C(2) = 9 + 5 = 14 \\ V_5 + C(1) = 10 + 1 = 11 \\ V_6 = 17 \end{cases}$$

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17		

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17		

$$C(7) = \max \begin{cases} V_1 + C(6) = 1 + 17 = 18 \\ V_2 + C(5) = 5 + 13 = 18 \\ V_3 + C(4) = 8 + 10 = 18 \\ V_4 + C(3) = 9 + 8 = 17 \\ V_5 + C(2) = 10 + 5 = 15 \\ V_6 + C(1) = 17 + 1 = 18 \\ V_7 = 17 \end{cases}$$

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	

$$C(8) = \max \begin{cases} V_1 + C(7) = 1 + 18 = 19 \\ V_2 + C(6) = 5 + 17 = 22 \\ V_3 + C(5) = 8 + 13 = 21 \\ V_4 + C(4) = 9 + 10 = 19 \\ V_5 + C(3) = 10 + 8 = 18 \\ V_6 + C(2) = 17 + 5 = 22 \\ V_7 + C(1) = 17 + 1 = 18 \\ V_8 = 20 \end{cases}$$

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	22

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

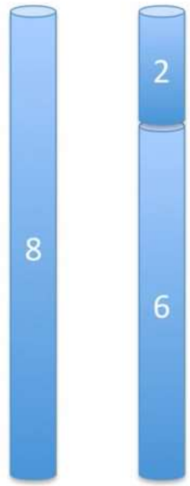
Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	22

Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	22

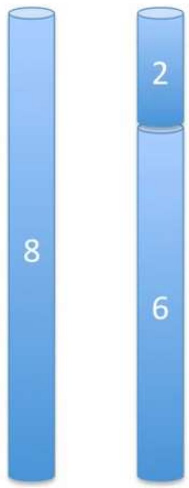
Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

$$C(6) = V_6 = 17$$



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	22

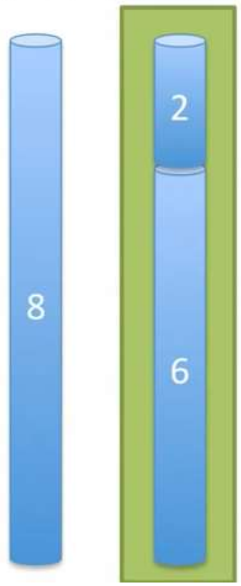
Example

Rod Cutting DP

$$C(i) = \max_{1 \le k \le i} \{V_k + C(i-k)\}$$

$$C(8) = V_2 + C(6) = 5 + 17 = 22$$

$$C(6) = V_6 = 17$$



Length	1	2	3	4	5	6	7	8
Price $	1	5	8	9	10	17	17	20

C(i)

Len (i)	1	2	3	4	5	6	7	8
Opt	1	5	8	10	13	17	18	22

The optimal way to cut a rod of length 8!

RoD Cutting DP Iterative Algorithms

ITERATIVE SOLUTION 1

```
RodCutting(n, Price[ ])
    allocate Table[0...n]
    Table[0...n] = 0
    for(length = 1; length ≤ n; length++)
        for(i = 1; i ≤ length; i++)
            tmp = Price[i] + Table[length−i]
            if(tmp > Table[length])
                Table[length] = tmp
    return Table[n]
```

Try it by yourself on the below data

Length	1	2	3	4	5	6	7	8
Price	2	5	9	10	12	13	15	16

ITERATIVE SOLUTION 2

```
RodCutting(n, Price[ ])
    allocate Table[0...n], Cuts[0...n]
    Table[0...n] = 0
    for(length = 1; length ≤ n; length++)
        for(i = 1; i ≤ length; i++)
            tmp = Price[i] + Table[length−i]
            if(tmp > Table[length])
                Table[length] = tmp
                Cuts[length] = i
    AnswerSet = {}
    while(n > 0)
        AnswerSet.add(Cuts[n])
        n −= Cuts[n]
    return AnswerSet
```

Number of cuts of rod to know with the update of optimal price update

Runtime: $\sum_{i=1}^{n} i = \Theta(n^2)$ (iterative or memoized)

Space: $\Theta(n)$ (iterative or memoized)

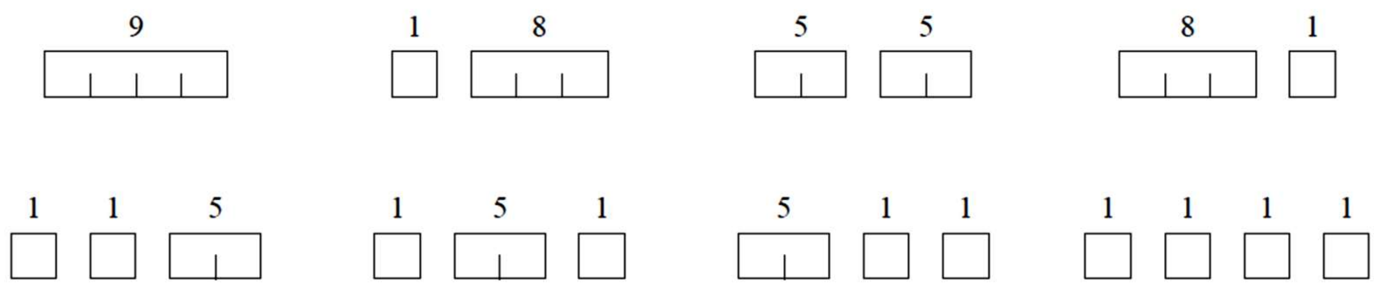
Time Complexity with/without DP

Without dynamic programming, the problem has a complexity of $O(2^n)$!

For a rod of length 8, there are 128 (or 2^{n-1}) ways to cut it!

With dynamic programming, and this top down approach, the problem is reduced to $O(n^2)$

Ex: a rod of length 4 (Summary)



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

i	r_i	optimal solution
1	1	1 (no cuts)
2	5	2 (no cuts)
3	8	3 (no cuts)
4	10	2 + 2
5	13	2 + 3
6	17	6 (no cuts)
7	18	1 + 6 or 2 + 2 + 3
8	22	2 + 6

Computing a binomial coefficient by DP

•Binomial coefficients are coefficients of the binomial formula:

$(a + b)^n = C(n,0)a^n b^0 + \dots + C(n, k)a^{n-k}b^k + \dots + C(n, n)a^0 b^n$

•Recurrence: $C(n, k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n, n) = 1$ for $n \geq 0$

Value of $C(n, k)$ can be computed by filling a table:

	0	1	2	. . .	k-1	k
0	1					
1	1	1				
.						
.						
.						
n-1					$C(n-1,k-1)$	$C(n-1,k)$
n						$C(n,k)$

Computing $C(n, k)$: pseudocode and analysis

ALGORITHM *Binomial*(n, k)

//Computes $C(n, k)$ by the dynamic programming algorithm
//Input: A pair of nonnegative integers $n \geq k \geq 0$
//Output: The value of $C(n, k)$
for $i \leftarrow 0$ **to** n **do**
  for $j \leftarrow 0$ **to** $\min(i, k)$ **do**
    if $j = 0$ **or** $j = i$
      $C[i, j] \leftarrow 1$
    else $C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$
return $C[n, k]$

Time efficiency: $\Theta(nk)$

Space efficiency: $\Theta(nk)$

Thank You!!!

Have a good day