COMP3600/6466 Algorithms

Lecture 10

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

Dynamic Programming: Summary

Two key ingredients:

1. Optimal substructure

2. Overlapping subproblems

Top-Down recursive memoisation

Easy and intuitive implementation

Can run out of memory due to nested calls

Bottom-up iterative procedure

Memory efficient

less intuitive implementation

Both have the same asymptotic bound

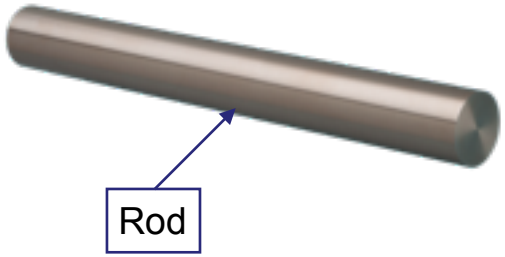
Dynamic Programming: How To

1. Characterise the optimal substructure or show that none exists
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution

Use memoization or bottom-up DP!

4. Construct an optimal solution from computed information

Rod Cutting Problem



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

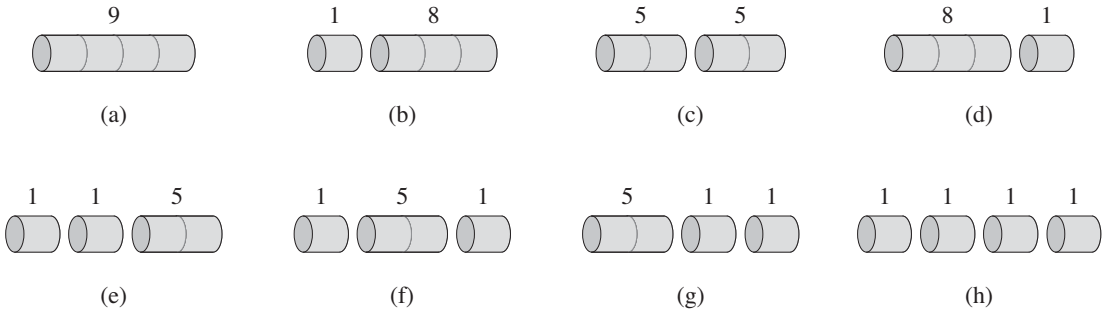


Figure 15.2 The 8 possible ways of cutting up a rod of length 4. Above each piece is the value of that piece, according to the sample price chart of Figure 15.1. The optimal strategy is part (c)—cutting the rod into two pieces of length 2—which has total value 10.

Rod Cutting Problem

1. Characterise the optimal substructure or show that none exists

The problem has an optimal substructure:
If the optimal solution for a rod of size n includes a cut at position i
then the cutting solution for the remaining sub-rod of size $n - i$ is optimal.

Optimal Substructure Proof: How To

Use a proof by contradiction:
1. Assume that the solution is composed of a non-optimal sub-solution
2. Show that this contradicts the optimality assumption of the global solution

(Hint: Use an optimal sub-solution to improve the overall solution)

The Cut and Paste method

The problem has an optimal substructure:
If the optimal solution for a rod of size n includes a cut at position i
then the cutting solution for the remaining sub-rod of size $n - i$ is optimal.

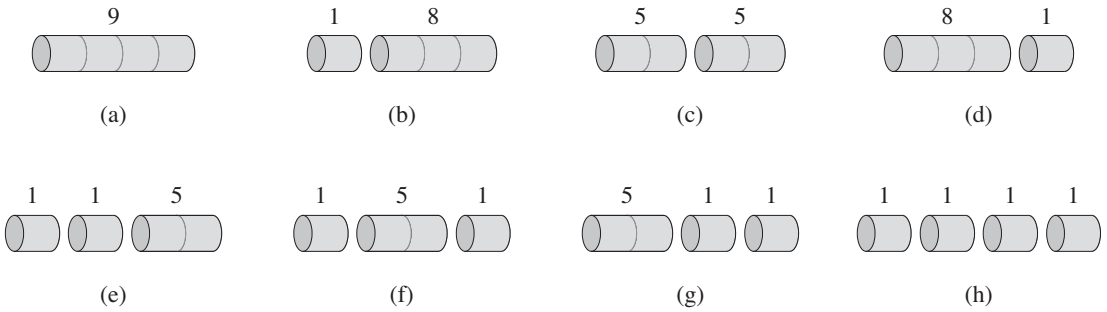
Proof.

1. Assume that the sub-rod of size $n - i$ is not cut optimally
2. If I change the cutting solution for the sub-rod with an optimal one, I can improve the cost of the overall solution, which contradicts the optimality assumption of the global cutting solution.

Rod Cutting Problem

2. Recursively define the value of an optimal solution

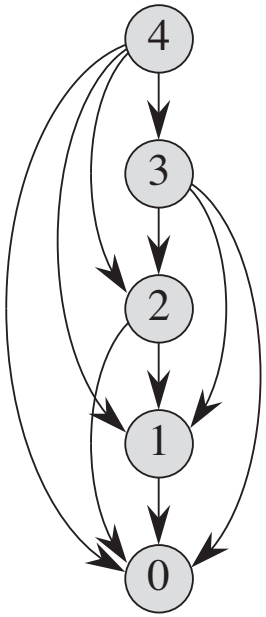
$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$



Rod Cutting Problem

3. Compute the value of an optimal solution

Use memoization or bottom-up DP!



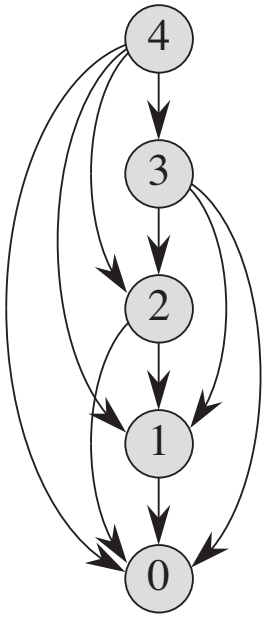
MEMOIZED-CUT-ROD-AUX(p, n, r)

```
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
```

Rod Cutting Problem

3. Compute the value of an optimal solution

Use memoization or 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]
```

$$\sum_{j=1}^{n} j = \frac{(n+1)n}{2} = \Theta(n^2)$$

Rod Cutting Problem

4. Construct an optimal solution from computed information

   At each iteration in the loop (recursion level), record the solution that leads to the corresponding optimal 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
```

Application 2: The Matrix

Matrix Mathematics Everywhere!

- Medicine: CAT scans and MRI's
- Computer Graphics: Reflection and Refraction
- Robotics: Calculate movements
- Physics: Electrical systems and quantum mechanics

What's a Matrix?

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

← second row

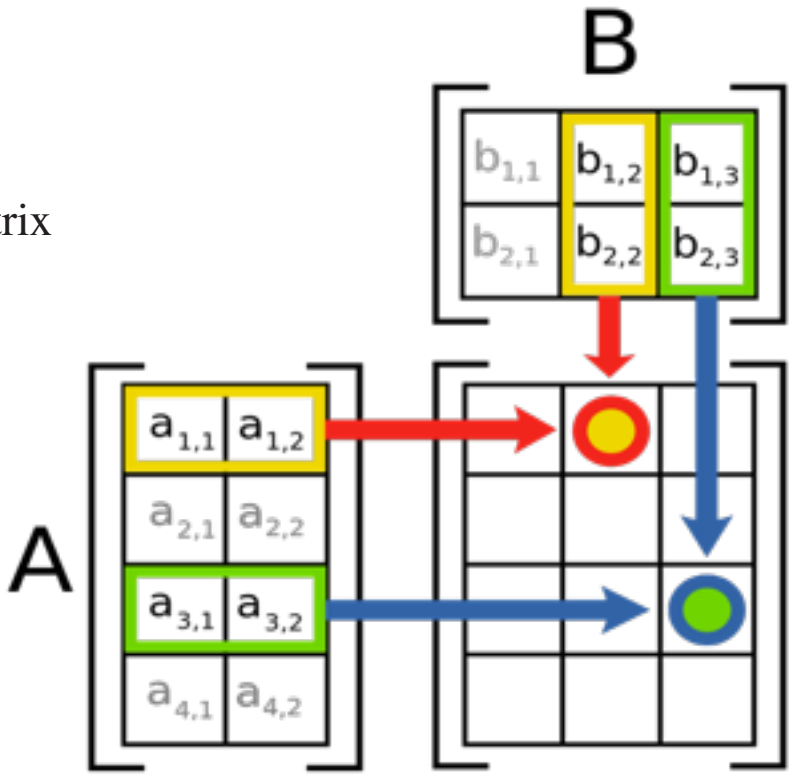
↑ third column

We say that A is $m \times n$ to indicate that it has m rows and n columns

Matrix Multiplication

MATRIX-MULTIPLY(A, B)

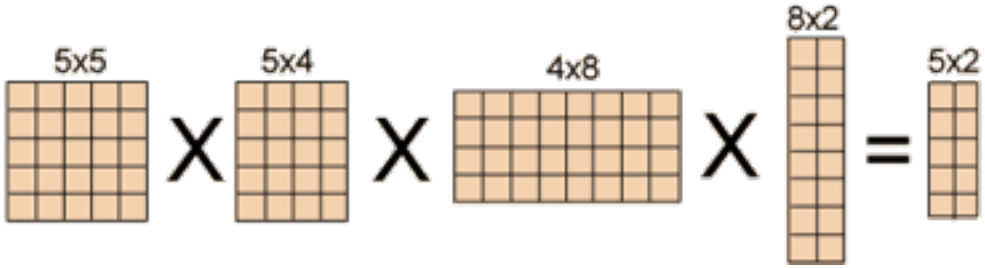
```
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7              for k = 1 to A.columns
8                  c_ij = c_ij + a_ik · b_kj
9      return C
```



Total number of scalar multiplication $= A.rows \times B.columns \times A.columns$

Matrix Chain Multiplication

$$A_1 \times A_2 \cdots \times A_n$$



The size of matrix A_i will be denoted $p_{i-1} \times p_i$

The size of the resulting matrix is $p_0 \times p_n$

Choice of Inner Multiplication

$n = 3$

$$((A_1 \cdot A_2) \cdot A_3)$$
$$(A_1 \cdot (A_2 \cdot A_3))$$

$n = 4$

$$((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$$
$$(A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$$
$$(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$$
$$(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$$
$$((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$$

Choice of parenthesisation or splitting

Does it make a difference?

$$n = 3$$

$$((A_1 \cdot A_2) \cdot A_3)$$

$$(A_1 \cdot (A_2 \cdot A_3))$$

1 hour of computation
vs
6 minutes!

Suppose that A_1 is 10×100, A_2 is 100×5, and A_3 is 5×50

First solution:
$A_1 \cdot A_2$ performs $10 \cdot 100 \cdot 5 = 5000$ scalar multiplications.
$A_{12} \cdot A_3$ performs $10 \cdot 5 \cdot 50 = 2500$ scalar multiplications.

Second solution:
$A_2 \cdot A_3$ performs $100 \cdot 5 \cdot 50 = 25,000$ scalar multiplications.
$A_1 \cdot A_{23}$ performs $10 \cdot 100 \cdot 50 = 50,000$ scalar multiplications.

$$7{,}500 \text{ vs } 75{,}000$$

Problem statement

Given the chain $A_1 \cdot A_2 \ldots \cdot A_n$, fully parenthesise/split it in a way that minimises the number of scalar multiplications.

$$((A_1 \cdot A_2) \cdot (A_3 \cdot A_4))$$
$$(A_1 \cdot (A_2 \cdot (A_3 \cdot A_4)))$$
$$(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$$
$$(((A_1 \cdot A_2) \cdot A_3) \cdot A_4)$$
$$((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$$

The size of matrix A_i will be denoted $p_{i-1} \times p_i$