VI. Dynamic Programming
Dynamic Programming – Overview

- Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
- Developed back in the day (1950s) when "programming" meant "tabular method" (like linear programming)
- Used for optimization problems
- Find a solution with the optimal value
- Minimization or maximization
Dynamic Programming – Overview

- Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
Dynamic Programming – Overview

- Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
- Developed back in the day (1950s) when “programming” meant “tabular method” (like linear programming)
Dynamic Programming – Overview

▶ Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
▶ Developed back in the day (1950s) when “programming” meant “tabular method” (like linear programming)
▶ Used for optimization problems
  ▶ Find a solution with the optimal value
  ▶ Minimization or maximization
Dynamic Programming

Four-step (two-phase) method:

1. Characterize the structure of an optimal solution
Dynamic Programming

Four-step (two-phase) method:

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
Dynamic Programming

Four-step (two-phase) method:

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
Dynamic Programming

Four-step (two-phase) method:
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
The rod cutting problem

Problem statement:

- **Input:**
  1) a rod of length $n$
  2) an array of prices $p_i$ for a rod of length $i$ for $i = 1, \ldots, n$.

- **Output:**
  1) the maximum revenue $r_n$ obtainable for a rod of length $n$
  2) optimal cut, if necessary.

In short, How to cut a rod into pieces in order to maximize the revenue you can get?
The rod cutting problem

Problem statement:

▶ Input:
  1) a rod of length \( n \)
  2) an array of prices \( p_i \) for a rod of length \( i \) for \( i = 1, \ldots, n \).

▶ Output:
  1) the maximum revenue \( r_n \) obtainable for a rod of length \( n 
  2) optimal cut, if necessary.
The rod cutting problem

Problem statement:

► **Input:**
   1) a rod of length \( n \)
   2) an array of prices \( p_i \) for a rod of length \( i \) for \( i = 1, \ldots, n \).

► **Output:**
   1) the maximum revenue \( r_n \) obtainable for a rod of length \( n \)
   2) optimal cut, *if necessary.*

In short,

*How to cut a rod into pieces in order to maximize the revenue you can get?*
The rod cutting problem

Example

<table>
<thead>
<tr>
<th>rod length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
## The rod cutting problem

### Example

<table>
<thead>
<tr>
<th>rod length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>$r_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>17</td>
<td>18</td>
<td>22</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

- $r_i$: maximum revenue of a rod of length $i$
- $s_i$: optimal size of the first piece to cut
The rod cutting problem

A brute-force solution:

*cut up a rod of length* $n$ *in* $2^{n-1}$ *different ways*
The rod cutting problem

A **brute-force** solution:

\[
\text{cut up a rod of length } n \text{ in } 2^{n-1} \text{ different ways}
\]

Cost: \( \Theta(2^{n-1}) \)
The rod cutting problem

Dynamic Programming – Phase I:

- Since every optimal solution \( r_n \) has a **leftmost** cut with length \( i \), the optimal revenue \( r_n \) is given by

\[
    r_n = \max \{ p_1 + r_{n-1}, \ p_2 + r_{n-2}, \ldots, \ p_{n-1} + r_1, \ p_n + r_0 \} \\
    = \max_{1 \leq i \leq n} \{ p_i + r_{n-i} \} \quad (1)
\]
The rod cutting problem

Dynamic Programming – Phase I:

Since every optimal solution \( r_n \) has a leftmost cut with length \( i \), the optimal revenue \( r_n \) is given by

\[
\begin{align*}
r_n &= \max \{ p_1 + r_{n-1}, p_2 + r_{n-2}, \ldots, p_{n-1} + r_1, p_n + r_0 \} \\
&= \max_{1 \leq i \leq n} \{ p_i + r_{n-i} \} \\
&= p_{i^*} + r_{n-i^*}
\end{align*}
\]

where

\[
\begin{align*}
i^* &= \text{the index attains the maximum} \\
&= \text{the length of the leftmost cut}
\end{align*}
\]
The rod cutting problem

Dynamic Programming – Phase II:

- How to compute $r_n$ by the expression (1)
  - Recursive solution:
    - top-down, no memoization
    - Calling graph

![Calling graph diagram](image)

Cost: let $T(n)$ be the number of calls to compute $r_n$; then

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = \Theta(2^n)$$

for $n > 1$ and $T(0) = 1$. 
The rod cutting problem

Dynamic Programming – Phase II:

▶ How to compute $r_n$ by the expression (1)

▶ Recursive solution:

▶ top-down, no memoization

▶ Calling graph

▶ Cost: let $T(n)$ be the number of calls to compute $r_n$; then

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = \Theta(2^n) \quad \text{for } n > 1$$

and $T(0) = 1$. 
The rod cutting problem

Dynamic Programming – Phase II:

- How to compute $r_n$ by the expression (1), cont’d
  - Iterative solution
    - bottom-up, memoization (Pseudocode – see next page)
    - Calling graph
The rod cutting problem

Dynamic Programming – Phase II:

- How to compute $r_n$ by the expression (1), cont’d
  - Iterative solution
    - bottom-up, memoization (Pseudocode – see next page)
    - Calling graph

- Cost: $T(n) = \Theta(n^2)$
The rod cutting problem

cut-rod(p,n)
// an iterative (bottom-up) procedure for finding ‘‘r’’ and
// the optimal size of the first piece to cut off ‘‘s’’
Let r[0...n] and s[0...n] be new arrays
r[0] = 0
for j = 1 to n
    // find q = max{p[i]+r[j-i]} for 1 <= i <= j
    q = -infty
    for i = 1 to j
        if q < p[i] + r[j-i]
            q = p[i] + r[j-i]
            s[j] = i
        end if
    end for
    r[j] = q
end for
return r and s
The rod cutting problem

Example

<table>
<thead>
<tr>
<th>rod length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>$r_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>17</td>
<td>18</td>
<td>22</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

- $r_i$: maximum revenue of a rod of length $i$
- $s_i$: optimal size of the first piece to cut

Note: $s_i = i_*$ in expression (2).