Dynamic Programming

- 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
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

- Problem statement:

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

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

- Output: The maximum revenue r_n obtainable for rods whose length sum to n

Rod cutting

Example

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
r_i	1	5	8	10	13	17	18	22	25	30

- r_i: maximum revenue of a rod of length i

Rod cutting

A brute-force solution:

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

Cost: $\Theta(2^{n-1})$

Rod cutting

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

$$
\begin{aligned}
r_n &= \max\{p_1 + r_{n-1},\ p_2 + r_{n-2},\ \ldots,\ p_{n-1} + r_1,\ p_n\} \\
&= \max_{1 \le i \le n} \{p_i + r_{n-i}\} \qquad (1) \\
&= p_{i_*} + r_{n-i_*} \qquad (2)
\end{aligned}
$$

  where

$$
\begin{aligned}
i_* &= \text{the index attains the maximum} \\
&= \text{the length of the leftmost cut}
\end{aligned}
$$

Rod cutting

Dynamic Programming – Phase II:

How to compute r_n by the expression (1) ?

1. Recursive solution:
   - top-down, no memoization
   - Cost:

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

2. Iterative solution
   - bottom-up, memoization
     Pseudocode – see next page
   - Cost:

$$T(n) = \Theta(n^2)$$

Rod cutting

Pseudocode

```
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
```

Rod cutting

Example

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
r_i	1	5	8	10	13	17	18	22	25	30
s_i	1	2	3	2	2	6	1	2	3	10

- 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).