Lecture 18: Elements of Dynamic Programming

COMS10007 - Algorithms

Dr. Christian Konrad

02.04.2019

Elements of Dynamic Programming

Solving a Problem with Dynamic Programming:

1. Identify optimal substructure
2. Give recursive solution
3. Compute optimal costs
4. Construct optimal solution

Discussion:

- Steps 1 and 2 requires studying the problem at hand
- Steps 3 and 4 are usually straightforward

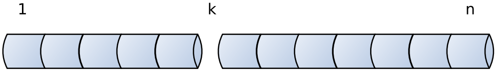
Step 1: Identify Optimal Substructure

Optimal Substructure Problem **P** exhibits *optimal substructure* if:

An optimal solution to **P** contains within it optimal solutions to subproblems of **P**.

Examples: Let OPT be optimal solution

- POLE-CUTTING: If OPT cuts at position k then cuts within $\{1, \dots, k-1\}$ form opt. solution to pole of len. k, and cuts within $\{k+1, \dots, n\}$ form opt. solution to pole of len. $n-k$.



- MATRIX-CHAIN-PARENTHESIZATION: If in OPT final multiplication is $A_{1k} \times A_{(k+1)n}$ then OPT contains optimal parenthesizations of $A_1 \times \dots \times A_k$ and $A_{k+1} \times \dots \times A_n$

$$(A_1 \times (A_2 \times A_3)) \times ((A_4 \times A_5) \times A_6)$$

Step 2. Give Recursive Solution

Define Table for Storing Optimal Solutions to Subproblems:
Optimal substructure indicates how subproblems look like

- POLE-CUTTING:
  OPT contains optimal solutions to shorter lengths
  $\rightarrow$ Store optimal solutions for every length in $\{1, \ldots, n\}$
  (table of length n)

- MATRIX-CHAIN-PARENTHESIZATION:
  OPT contains optimal parenthesizations for subproducts
  $A_i \times \cdots \times A_j$
  $\rightarrow$ Store optimal parenthesizations for every subproduct
  $A_i \times \cdots \times A_j$ (table of size n^2)

Step 2. Give Recursive Solution (2)

Express Optimal Solutions Recursively:

- POLE-CUTTING: (p_k: price for selling a pole of length k)

$m[i] :=$ maximum revenue to pole of length i

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

- MATRIX-CHAIN-PARENTHESIZATION:

$m[i,j] :=$ min. # scalar mult. to compute $A_i \times A_{i+1} \times \cdots \times A_j$

$$\begin{aligned} m[i,j] &= \min_{i \le k < j} m[i,k] + m[k+1,j] \\ &+ \text{“cost for computing } A_{ik} \times A_{(k+1)j}\text{”} \end{aligned}$$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
-	-	-	-	-	-	-	-	-	-	-

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
-	-	-	-	-	-	-	-	-	-	-

Initialize base cases: $m[0] = 0$ and $m[1] = p_1$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	-	-	-	-	-	-	-	-	-

Initialize base cases: $m[0] = 0$ and $m[1] = p_1$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	-	-	-	-	-	-	-	-	-

$m[2] = \max\{p_1 + m_1, p_2 + m_0\} = \max\{1+1, 5+0\} = 5$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	-	-	-	-	-	-	-	-

$m[2] = \max\{p_1 + m_1, p_2 + m_0\} = \max\{1 + 1, 5 + 0\} = 5$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	-	-	-	-	-	-	-	-

$m[3] = \max\{p_1 + m_2, p_2 + m_1, p_3 + m_0\} = \max\{1+5, 5+1, 8+0\} = 8$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	-	-	-	-	-	-	-

$$m[3] = \max\{p_1 + m_2, p_2 + m_1, p_3 + m_0\} = \max\{1+5, 5+1, 8+0\} = 8$$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	-	-	-	-	-	-	-

$m[4] = \max\{p_1 + m_3, p_2 + m_2, p_3 + m_1, p_4 + m_0\} = \max\{1 + 8, 5 + 5, 8 + 1, 9\} = 10$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	-	-	-	-	-	-

$m[4] = \max\{p_1 + m_3, p_2 + m_2, p_3 + m_1, p_4 + m_0\} = \max\{1 + 8, 5 + 5, 8 + 1, 9\} = 10$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	-	-	-	-	-	-

$m[5] = \max\{p_1 + m_4, p_2 + m_3, p_3 + m_2, p_4 + m_1, p_5 + m_0\} = \max\{1 + 10, 5 + 8, 8 + 2, 9 + 1, 10\} = 13$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	13	-	-	-	-	-

$m[5] = \max\{p_1 + m_4, p_2 + m_3, p_3 + m_2, p_4 + m_1, p_5 + m_0\} = \max\{1 + 10, 5 + 8, 8 + 2, 9 + 1, 10\} = 13$

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for Pole-Cutting

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	13	-	-	-	-	-

...

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \le k \le i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	13	17	18	22	25	30

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	13	17	18	22	25	30

The maximum revenue obtainable for a pole of length 10 is 30

Compute Optimal Costs

Two Possibilities:

- Bottom-up
- Top-down with memoization

Example: Bottom-up for POLE-CUTTING

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

$$m[i] = \max_{1 \leq k \leq i} p_k + m_{i-k}$$

0	1	2	3	4	5	6	7	8	9	10
0	1	5	8	10	13	17	18	22	25	30

But how can we find out how to cut the pole?

Step 4: Construct Optimal Solution

Keep Track of Optimal Choices: store optimal choices in array s

Require: Integer n, array p of length n with prices
  Let $r[0 \dots n]$ be a new array
  $r[0] \leftarrow 0$
  for $j = 1 \dots n$ **do**
    $r[j] \leftarrow -\infty$
    for $i = 1 \dots j$ **do**
      $r[j] \leftarrow \max\{r[j], p[i] + r[j-i]\}$
  return $r[n]$

Algorithm $\textsc{Bottom-Up-Cut-Pole}(p, n)$

- $s[i]$ contains position of first cut in optimal solution
- Easy to reconstruct all cuts

Step 4: Construct Optimal Solution

Keep Track of Optimal Choices: store optimal choices in array s

Require: Integer n, array p of length n with prices
  Let $r[0 \dots n]$ be a new array, let $s[1 \dots n]$ be a new array
  $r[0] \leftarrow 0$
  for $j = 1 \dots n$ **do**
    $r[j] \leftarrow -\infty$
    for $i = 1 \dots j$ **do**
      if $p[i] + r[j-i] > q$ **then**
        $r[j] \leftarrow p[i] + r[j-i]$
        $s[j] \leftarrow i$
  return $r[n]$

Algorithm BOTTOM-UP-CUT-POLE(p, n)

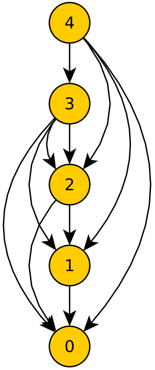
- $s[i]$ contains position of first cut in optimal solution
- Easy to reconstruct all cuts

Subproblem Graph and Runtime

Subproblem Graph

- One node for each subproblem
- Directed edge from a subproblem A to subproblem B if the solution of A depends on the solution of B

Example: POLE-CUTTING



Runtime of Dynamic Programming Algorithm:

- Total number of subproblems t
- Maximum number of subproblems a subproblem depends on s
- Runtime: $O(s \cdot t)$ (assuming that computing solution takes time $O(s)$)