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

Review: the rod cutting problem

Dynamic Programming Solution

- Phase I:
  Since every optimal solution r_n has a leftmost cut with length i, the optimal revenue r_n is given by

$$r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\} = p_{i_*} + r_{n-i_*}$$

- Phase II:
  compute r_n in bottom-up iteration (memoization)

Matrix-chain multiplication – DP case study 2

Review: Matrix-matrix multiplication

- Given A of order $p \times q$ and B of order $q \times r$, then $C = AB$ is of order $p \times r$, and (i, j)-entry of C is given by

$$C_{ij} = \sum_{k=1}^{q} A_{ik} B_{kj}$$

- Cost: pqr scalar multiplications

Matrix-chain multiplication

Review: ordering of matrix-chain multiplication

- Given A_1 of order $p_0 \times p_1$
  A_2 of order $p_1 \times p_2$
  A_3 of order $p_2 \times p_3$
  Then different orderings of the product $A_1 A_2 A_3$ generate the same result
  $$(A_1 A_2)A_3 = A_1(A_2 A_3),$$
  but the costs are different!

- Example:
  $$A_1(10 \times 5), \quad A_2(5 \times 10), \quad A_3(10 \times 5)$$
  - *cost of* $(A_1 A_2)A_3 = 10 \cdot 5 \cdot 10 + 10 \cdot 10 \cdot 5 = 1000$
  - *cost of* $A_1(A_2 A_3) = 5 \cdot 10 \cdot 5 + 10 \cdot 5 \cdot 5 = 500$

Matrix-chain multiplication

Problem statement:

Input: *A sequence (chain) of* $(A_1, A_2, \ldots, A_n)$ *of matrices, where* A_i *is of order* $p_{i-1} \times p_i$.

Matrix-chain multiplication

Problem statement:

Input: *A sequence (chain) of* $(A_1, A_2, \ldots, A_n)$ *of matrices, where* A_i *is of order* $p_{i-1} \times p_i$.

Output: *full parenthesization (ordering) for the product* $A_1 \cdot A_2 \cdot \cdots \cdot A_n$ *that minimizes the number of (scalar) multiplications.*

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering
- Counting the total number of orderings

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering
- Counting the total number of orderings
  1. Define
     $P(n)$ = the number of orderings for a chain of n matrices

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering
- Counting the total number of orderings

1. Define
   $P(n) =$ the number of orderings for a chain of n matrices
2. Then $P(1) = 1$ and for $n \geq 2$,

$$
\begin{aligned}
P(n) &= P(1)P(n-1) + P(2)P(n-2) + \cdots + P(n-1)P(1) \\
&= \sum_{k=1}^{n-1} P(k)P(n-k)
\end{aligned}
$$

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering
- Counting the total number of orderings
  1. Define
     $P(n) =$ the number of orderings for a chain of n matrices
  2. Then $P(1) = 1$ and for $n \geq 2$,
     $$\begin{aligned} P(n) &= P(1)P(n-1) + P(2)P(n-2) + \cdots + P(n-1)P(1) \\ &= \sum_{k=1}^{n-1} P(k)P(n-k) \end{aligned}$$
  3. $P(n)$ is called a *Catalan number*, which grows as $P(n) = \Omega(2^n)$

Matrix-chain multiplication

Brute-force solution

- Exhaustive search for determining the optimal ordering
- Counting the total number of orderings
  1. Define
     $P(n) =$ the number of orderings for a chain of n matrices
  2. Then $P(1) = 1$ and for $n \geq 2$,
     $$\begin{aligned} P(n) &= P(1)P(n-1) + P(2)P(n-2) + \cdots + P(n-1)P(1) \\ &= \sum_{k=1}^{n-1} P(k)P(n-k) \end{aligned}$$
  3. $P(n)$ is called a *Catalan number*, which grows as $P(n) = \Omega(2^n)$
- Therefore, exhaustive search for determining the optimal ordering is infeasible!

Matrix-chain multiplication

DP – step 1: *characterize the structure of an optimal ordering*

[1] Why? simply argue by contradiction: If there were a less costly way to order the product $A_1 \cdots A_k$, substituting that ordering within this (global) optimal ordering would produce another ordering of $A_1 A_2 \cdots A_n$, whose cost would be less than the optimum, a contradiction!

Matrix-chain multiplication

DP – step 1: *characterize the structure of an optimal ordering*

- An optimal ordering of the product $A_1 A_2 \cdots A_n$ splits the product between A_k and A_{k+1} for some k:

$$A_1 A_2 \cdots A_n = \underline{A_1 \cdots A_k} \cdot \underline{A_{k+1} \cdots A_n}$$

[1] Why? simply argue by contradiction: If there were a less costly way to order the product $A_1 \cdots A_k$, substituting that ordering within this (global) optimal ordering would produce another ordering of $A_1 A_2 \cdots A_n$, whose cost would be less than the optimum, a contradiction!

Matrix-chain multiplication

DP – step 1: *characterize the structure of an optimal ordering*

- An optimal ordering of the product $A_1 A_2 \cdots A_n$ splits the product between A_k and A_{k+1} for some k:

$$A_1 A_2 \cdots A_n = \underline{A_1 \cdots A_k} \cdot \underline{A_{k+1} \cdots A_n}$$

- Key observation: the ordering of $A_1 \cdots A_k$ within this ("global") optimal ordering must be an optimal ordering of (sub-product) $A_1 \cdots A_k$. [1]

[1] Why? simply argue by contradiction: If there were a less costly way to order the product $A_1 \cdots A_k$, substituting that ordering within this (global) optimal ordering would produce another ordering of $A_1 A_2 \cdots A_n$, whose cost would be less than the optimum, a contradiction!

Matrix-chain multiplication

DP – step 1: *characterize the structure of an optimal ordering*

- An optimal ordering of the product $A_1 A_2 \cdots A_n$ splits the product between A_k and A_{k+1} for some k:

$$A_1 A_2 \cdots A_n = \underline{A_1 \cdots A_k} \cdot \underline{A_{k+1} \cdots A_n}$$

- Key observation: the ordering of $A_1 \cdots A_k$ within this ("global") optimal ordering must be an optimal ordering of (sub-product) $A_1 \cdots A_k$. [1]
- Similar observation holds for $A_{k+1} \cdots A_n$
- Thus, an optimal ("global") solution contains within it the optimal ("local") solutions to subproblems. (**the optimal substructure property**)

[1] Why? simply argue by contradiction: If there were a less costly way to order the product $A_1 \cdots A_k$, substituting that ordering within this (global) optimal ordering would produce another ordering of $A_1 A_2 \cdots A_n$, whose cost would be less than the optimum, a contradiction!

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

- Define

$$m[i,j] = \text{min. number of multip. needed to compute } A_i \cdots A_j.$$

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

- Define

$$m[i,j] = \text{min. number of multip. needed to compute } A_i \cdots A_j.$$

- By the definition,

$$m[1,n] = \text{the cheapest way for the product } A_1 A_2 \cdots A_n.$$

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

- Define

$$m[i,j] = \text{min. number of multip. needed to compute } A_i \cdots A_j.$$

- By the definition,

$$m[1,n] = \text{the cheapest way for the product } A_1 A_2 \cdots A_n.$$

- $m[i,j]$ can be defined recursively

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

- Define

$$m[i,j] = \text{min. number of multip. needed to compute } A_i \cdots A_j.$$

- By the definition,

$$m[1,n] = \text{the cheapest way for the product } A_1 A_2 \cdots A_n.$$

- $m[i,j]$ can be defined recursively

  for $1 \leq i \leq j \leq n$,

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

Matrix-chain multiplication

DP – step 2: *recursively define the value of an optimal solution*

- Define

  $$m[i,j] = \text{min. number of multip. needed to compute } A_i \cdots A_j.$$

- By the definition,

  $$m[1,n] = \text{the cheapest way for the product } A_1 A_2 \cdots A_n.$$

- $m[i,j]$ can be defined recursively

  for $1 \le i \le j \le n$,

  $$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

- To construct an optimal ordering, we track

  the value k such that $m[i,j]$ attains the minimum $\equiv k_* \equiv s[i,j]$

Matrix-chain multiplication

DP – step 3: *compute the value of an optimal solution in a bottom-up approach*

- Compute $m[i, j]$ and $s[i, j]$ in a bottom-up approach. (see the pseudocode in next page)

Matrix-chain multiplication

DP – step 3: *compute the value of an optimal solution in a bottom-up approach*

- Compute $m[i, j]$ and $s[i, j]$ in a bottom-up approach. (see the pseudocode in next page)

- Cost: $T(n) = \Theta(n^3)$ since

Matrix-chain multiplication

DP – step 3: *compute the value of an optimal solution in a bottom-up approach*

- Compute $m[i,j]$ and $s[i,j]$ in a bottom-up approach. (see the pseudocode in next page)

- Cost: $T(n) = \Theta(n^3)$ since
  1. compute $n(n-1)/2$ entries of m-table
  2. for each entry of m-table, it finds the minimum of fewer than n expressions.

Matrix-chain multiplication

```
matrix-chain-order(p)
create m[1...n,1...n] and s[1...n,1...n] and n = length(p)-1
for i = 1 to n
  m[i,i] = 0
for d = 2 to n
  for i = 1 to n-d+1
     j = i + d - 1
     m[i,j] = +infty      //compute m[i,j]=min_k{...}
     for k = i to j-1
        q  = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j]
        if q < m[i,j]
           m[i,j] = q
           s[i,j] = k
        endif
     endfor
  endfor
endfor
return m and s
```

Matrix-chain multiplication

DP – step 4: *construct an optimal solution from computed m and s tables*

Matrix-chain multiplication

Example 1. Let `p = [3 1 4 5 4]`,
namely, A_1 is 3×1, A_2 is 1×4, A_3 is 4×5, A_4 is 5×4.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 12 35 52 ]          s = [ 0 1 1 1 ]
    [ 0  0 20 40 ]              [ 0 0 2 3 ]
    [ 0  0  0 80 ]              [ 0 0 0 3 ]
    [ 0  0  0  0 ]              [ 0 0 0 0 ]
```

Matrix-chain multiplication

Example 1. Let `p = [3 1 4 5 4]`,
namely, A_1 is 3×1, A_2 is 1×4, A_3 is 4×5, A_4 is 5×4.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 12 35 52 ]          s = [ 0 1 1 1 ]
    [ 0  0 20 40 ]              [ 0 0 2 3 ]
    [ 0  0  0 80 ]              [ 0 0 0 3 ]
    [ 0  0  0  0 ]              [ 0 0 0 0 ]
```

By m-table, the minimum number of multiplications is

```
m[1,4] = 52
```

Matrix-chain multiplication

Example 1. Let `p = [3 1 4 5 4]`,
namely, A_1 is 3×1, A_2 is 1×4, A_3 is 4×5, A_4 is 5×4.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 12 35 52 ]        s = [ 0 1 1 1 ]
    [ 0  0 20 40 ]            [ 0 0 2 3 ]
    [ 0  0  0 80 ]            [ 0 0 0 3 ]
    [ 0  0  0  0 ]            [ 0 0 0 0 ]
```

By `m`-table, the minimum number of multiplications is

`m[1,4] = 52`

By `s`-table, an optimal parenthesization (ordering) of the matrix-chain multiplication is given by

$$(A_1)((A_2 A_3) A_4)$$

Matrix-chain multiplication

Example 2. Let `p = [30 35 15 5 10 20 25]`.

Matrix-chain multiplication

Example 2. Let `p = [30 35 15 5 10 20 25]`.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 15750 7875 9375 11875 15125 ]   s = [ 0 1 1 3 3 3 ]
    [ 0     0 2625 4375  7125 10500 ]       [ 0 0 2 3 3 3 ]
    [ 0     0    0  750  2500  5375 ]       [ 0 0 0 3 3 3 ]
    [ 0     0    0    0  1000  3500 ]       [ 0 0 0 0 4 5 ]
    [ 0     0    0    0     0  5000 ]       [ 0 0 0 0 0 5 ]
    [ 0     0    0    0     0     0 ]       [ 0 0 0 0 0 0 ]
```

Matrix-chain multiplication

Example 2. Let `p = [30 35 15 5 10 20 25]`.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 15750 7875 9375 11875 15125 ]   s = [ 0 1 1 3 3 3 ]
    [ 0     0 2625 4375  7125 10500 ]       [ 0 0 2 3 3 3 ]
    [ 0     0    0  750  2500  5375 ]       [ 0 0 0 3 3 3 ]
    [ 0     0    0    0  1000  3500 ]       [ 0 0 0 0 4 5 ]
    [ 0     0    0    0     0  5000 ]       [ 0 0 0 0 0 5 ]
    [ 0     0    0    0     0     0 ]       [ 0 0 0 0 0 0 ]
```

By `m`-table, the minimum number of multiplications is

```
m[1,6] = 15125
```

Matrix-chain multiplication

Example 2. Let `p = [30 35 15 5 10 20 25]`.

`matrix-chain-order(p)` generates the following m-table for optimal costs, and s-table for orderings:

```
m = [ 0 15750 7875 9375 11875 15125 ]   s = [ 0 1 1 3 3 3 ]
    [ 0     0 2625 4375  7125 10500 ]       [ 0 0 2 3 3 3 ]
    [ 0     0    0  750  2500  5375 ]       [ 0 0 0 3 3 3 ]
    [ 0     0    0    0  1000  3500 ]       [ 0 0 0 0 4 5 ]
    [ 0     0    0    0     0  5000 ]       [ 0 0 0 0 0 5 ]
    [ 0     0    0    0     0     0 ]       [ 0 0 0 0 0 0 ]
```

By `m`-table, the minimum number of multiplications is

`m[1,6] = 15125`

By `s`-table, an optimal parenthesization (ordering) of the matrix-chain multiplication is given by

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