Algorithms Theory

14 – Dynamic Programming (2)

Matrix-chain multiplication

Prof. Dr. S. Albers

Optimal substructure

Dynamic programming is typically applied to
optimization problems.

An optimal solution to the original problem contains
optimal solutions to smaller subproblems.

Matrix-chain multiplication

Given: sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of matrices

Goal: compute the product $A_1 \cdot A_2 \cdot \ldots . \cdot A_n$

Problem: Parenthesize the product in a way that minimizes the number of scalar multiplications.

Definition: A product of matrices is *fully parenthesized* if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses.

Examples of fully parenthesized matrix products of the chain $\langle A_1, A_2, \ldots, A_n \rangle$

All possible fully parenthesized matrix products of the chain $\langle A_1, A_2, A_3, A_4 \rangle$ are:

$$(A_1(A_2(A_3A_4)))$$

$$(A_1((A_2A_3)A_4))$$

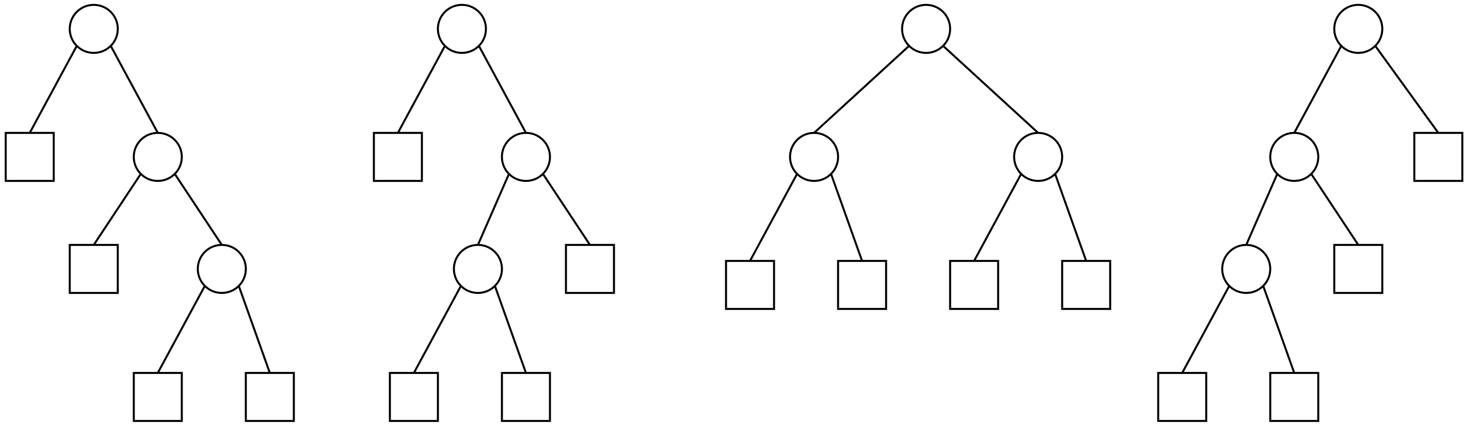
$$((A_1A_2)(A_3A_4))$$

$$((A_1(A_2A_3))A_4)$$

$$(((A_1A_2)A_3)A_4)$$

Number of different parenthesizations

Different parenthesizations correspond to different trees:



Number of different parenthesizations

Let *P(n)* be the number of alternative parenthesizations of the product $A_1 \ldots A_k A_{k+1} \ldots A_n$.

$$P(1) = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k) P(n-k) \quad \text{for } n \geq 2$$

$$P(n+1) = \frac{1}{n+1} \binom{2n}{n} \approx \frac{4^n}{n\sqrt{\pi n}} + O\left(\frac{4^n}{\sqrt{n^5}} \right)$$

$$P(n+1) = C_n \quad n - \text{th Catalan number}$$

Remark: Determining the optimal parenthesization by exhaustive search is not reasonable.

Multiplying two matrices

$$A = (a_{ij})_{p\times q},\ B = (b_{ij})_{q\times r},\ A\cdot B = C = (c_{ij})_{p\times r},$$

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}.$$

Algorithm *Matrix-Mult*
Input: $(p \times q)$ matrix A, $(q \times r)$ matrix B
Output: $(p \times \mathrm{r})$ matrix $C = A \cdot B$

```
for i := 1 to p do
    for j :=1 to r do
        C[i, j] := 0
        for k := 1 to q do
            C[i, j] := C[i, j] + A[i, k] ·B[k,j]
```

Number of multiplications and additions: $p \cdot q \cdot r$

Remark: Using this algorithm, multiplying two $(n \times n)$ matrices requires n^3 multiplications. This can also be done using $O(n^{2.376})$ multiplications.

Matrix-chain multiplication: Example

Computation of the product $A_1 A_2 A_3$, where

A_1 : (10 × 100) matrix
A_2 : (100 × 5) matrix
A_3 : (5 × 50) matrix

a) Parenthesization $((A_1 A_2) A_3)$ requires

$A' = (A_1 A_2)$:

$A' A_3$:

Sum:

Matrix-chain multiplication: Example

A_1 : (10×100) matrix
A_2 : (100×5) matrix
A_3 : (5×50) matrix

a) Parenthesization $(A_1 (A_2 A_3))$ requires

$A'' = (A_2 A_3)$:

$A_1 A''$:

Sum:

Structure of an optimal parenthesization

$(A_{i...j}) = ((A_{i...k})\ (A_{k+1....j}))$ $i \leq k < j$

Any optimal solution to the matrix-chain multiplication problem contains optimal solutions to subproblems.

Determining an optimal solution recursively:

Let m[i,j] be the minimum number of operations needed to compute the product $A_{i...j}$:

m[i,j] = 0, if $i = j$

m[i,j] = $\min_{i \leq k < j} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \}$, otherwise

s[i,j] = optimal splitting value *k*, i.e. the optimal parenthesization of $(A_{i...j})$ splits the product between A_k and A_{k+1}

Recursive matrix-chain multiplication

Algorithm *rec-mat-chain*(p, i, j)
Input: sequence $p = \langle p_0, p_1, ..., p_n \rangle$,
where $(p_{i-1} \times p_i)$ is the dimensionen of matrix A_i
Invariant: rec-mat-chain(p, i, j) returns $m[i, j]$

if $i = j$ **then return** 0
$m[i, j] := \infty$
for $k := i$ **to** $j - 1$ **do**
    $m[i, j] :=$ min($m[i,j]$, $p_{i-1}\, p_k\, p_j$ +
        rec-mat-chain(p, i, k) +
        rec-mat-chain(p, k+1, j))
return $m[i, j]$

Initial call: rec-mat-chain(p,1, n)

Recursive matrix-chain multiplication: Running time

Let *T*(*n*) be the time taken by rec-mat-chain(*p*,1,*n*).

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} \big(T(k) + T(n-k) + 1\big)$$

$$\geq n + 2\sum_{i=1}^{n-1} T(i)$$

$$\Rightarrow T(n) \geq 3^{n-1} \quad \text{(induction)}$$

Exponential running time!

Matrix-chain multiplication using dynamic programming

Algorithm *dyn-mat-chain*

Input: sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$, $(p_{i-1} \times p_i)$ the dimension of matrix A_i

Output: $m[1,n]$

n := length(p) – 1
for i := 1 **to** n **do** $m[i, i]$:= 0
for l := 2 **to** n **do** /* l = length of the subproblem */
  for i := 1 **to** $n - l + 1$ **do** /* i is the left index */
    $j := i + l - 1$ /* j is the right index */
    $m[i, j] := \infty$
    for $k := i$ **to** $j - 1$ **do**
      $m[i, j] := \min(m[i, j], \; p_{i-1} p_k p_j + m[i, k] + m[k + 1, j])$
return $m[1, n]$

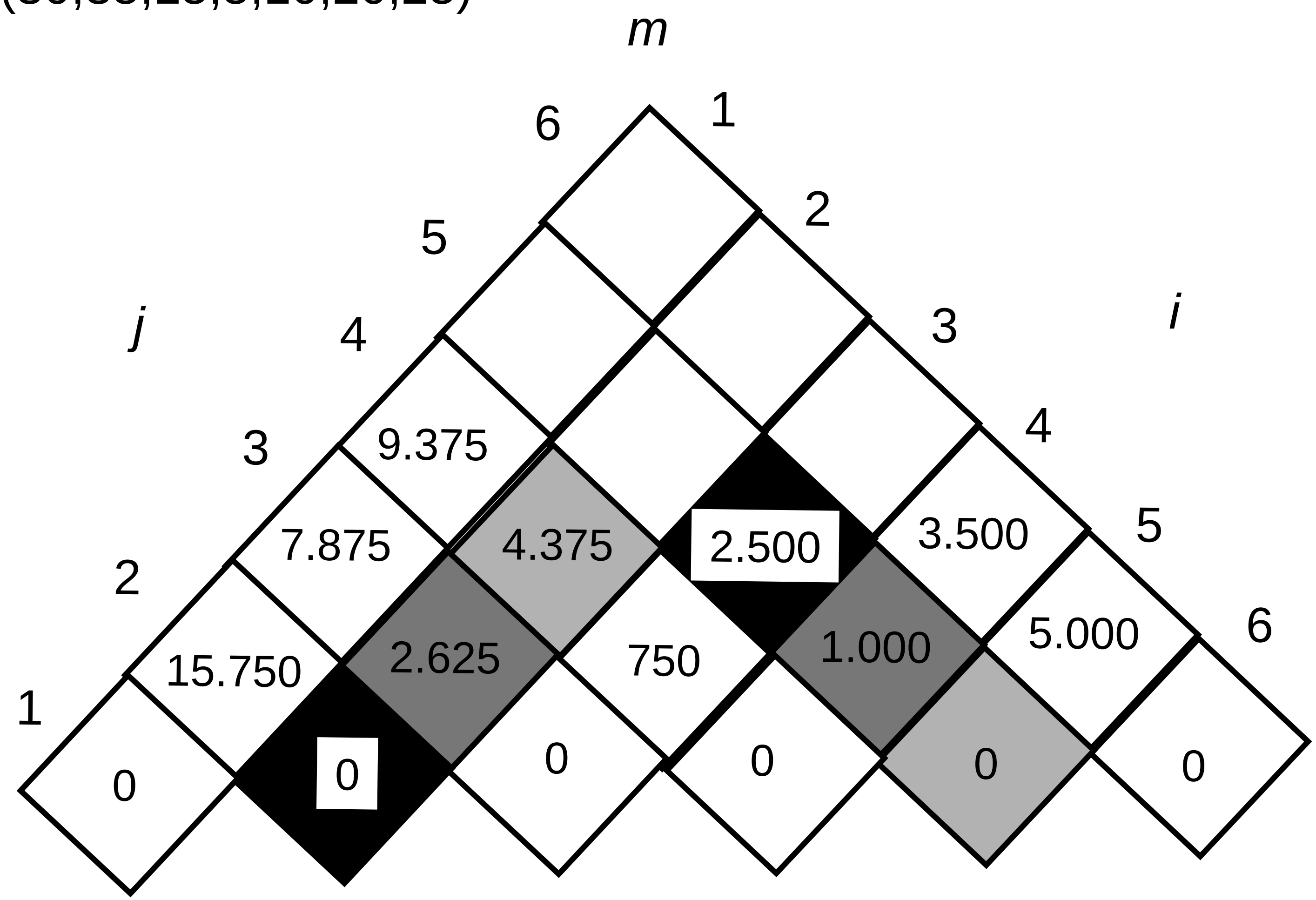
Example

A_1 (30×35)	A_4 (5×10)
A_2 (35×15)	A_5 (10×20)
A_3 (15×5)	A_6 (20×25)

$p = (30,35,15,5,10,20,25)$

Example

$p = (30,35,15,5,10,20,25)$



Example

$$m[2,5] =$$

$$\min_{2 \le k < 5}\left(m[2,k] + m[k+1,5] + p_1 p_k p_5\right)$$

$$\min\begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 \end{cases}$$

$$\min\begin{cases} 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

$$= 7125$$

Matrix-chain multiplication and optimal splitting values using dynamic programming

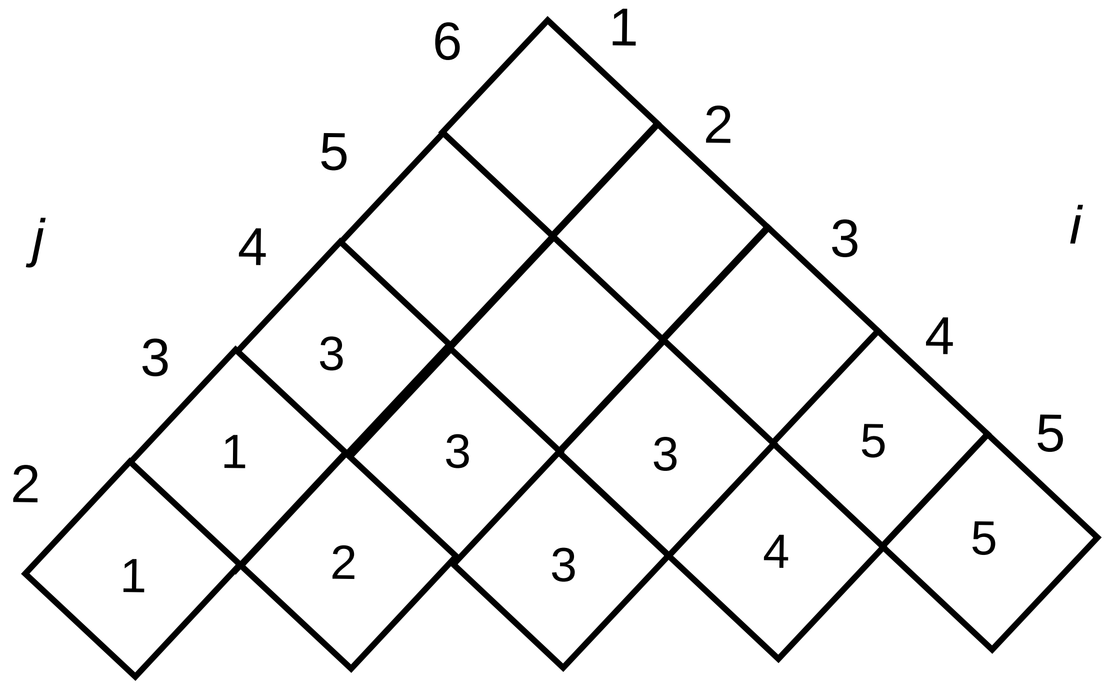
Algorithm *dyn-mat-chain(p)*

Input: sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$, $(p_{i-1} \times p_i)$ the dimension of matrix A_i

Output: $m[1,n]$ and a matrix $s[i,j]$ containing the optimal splitting values

```
n := length(p) – 1
for i := 1 to n do m[i, i] := 0
for l := 2 to n do
    for i := 1 to n – l +1 do
        j := i + l – 1
        m[i, j] := ∞
        for k := i to j – 1 do
            q := m[i, j]
            m[i, j] := min( m[i, j], p_{i-1} p_k p_j + m[i, k] + m[k + 1, j] )
            if m[i, j] < q then s[i, j] := k
return (m[1, n], s)
```

Example of splitting values



Computation of an optimal parenthesization

Algorithm *Opt-Parens*

Input: chain A of matrices, matrix s containing the optimal splitting values, two indices i and j

Output: an optimal parenthesization of $A_{i...j}$

if $i < j$
    then X := *Opt-Parens*(A, s, i, $s[i, j]$)
        Y := *Opt-Parens*(A, s, $s[i, j] + 1$, j)
        return $(X \cdot Y)$
    else return A_i

Initial call: *Opt–Parens*(A, s, 1, n)

Matrix-chain multiplication using dynamic programming (top-down approach)

„Memoization“ for increasing the efficiency of a recursive solution:

Only the *first time* a subproblem is encountered, its solution is computed and then stored in a table. Each subsequent time that the subproblem is encountered, the value stored in the table is simply looked up and returned (without repeated computation!).

Memoized matrix-chain multiplication („notepad method“)

Algorithm *mem-mat-chain*(p, i, j)

Invariant: mem-mat-chain(p, i, j) returns $m[i, j]$;
the value is correct if $m[i, j] < \infty$

if $i = j$ **then return** 0
if $m[i, j] < \infty$ **then return** $m[i, j]$
for $k := i$ **to** $j - 1$ **do**
    $m[i, j] := \min(m[i, j],\ p_{i-1}\, p_k\, p_j +$
        mem-mat-chain(p, i, k) +
        mem-mat-chain(p, $k + 1$, j))
return $m[i, j]$

Memoized matrix-chain multiplication

Call:

```
n := length(p) – 1
for i := 1 to n do
    for j := 1 to n do
        m[i, j] := ∞
mem-mat-ket(p,1,n)
```

The computation of all entries *m*[*i, j*] using *mem-mat-chain* takes $O(n^3)$ time.

$O(n^2)$ entries

each entry *m*[*i, j*] is computed once

each entry *m*[*i, j*] is looked up during the computation of *m*[*i´,j´*] if
i´= i and *j´>j* or *j´= j* and *i´< i*

→ *m*[*i, j*] is looked up during the computation of at most 2*n* entries

Remarks about matrix-chain multiplication

1. There is an algorithm that determines an optimal parenthesization in time $O(n \log n)$.

2. There is a linear time algorithm that determines a parenthesization using most $1.155 \cdot M_{opt}$ multiplications.