Algorithms

Kiho Choi

Fall, 2022

Department of AI·Software
Gachon University

3. Dynamic Programming III

Contents

- Matrix chain order
- Implementation on Matrix chain order

Matrix-chain multiplication

Given a $p \times q$ matrix A and a $q \times r$ matrix B, computing the product AB using the standard method requires a total of pqr scalar multiplications—q for each of the pr entries.

What if A is $p \times q$, B is $q \times r$, and C is $r \times s$?

Depends on how we use associativity:

- $(AB)C$: $pqr + prs$
- $A(BC)$: $qrs + pqs$

If $p = r = 10$ and $q = s = 100$, this is 20,000 vs. 200,000 multiplications!

Full parenthesization

- 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.

- A_1
- $(A_1 A_2)$
- $(A_1(A_2 A_3))$, $((A_1 A_2)A_3)$
- $(A_1(A_2(A_3 A_4)))$, $(A_1((A_2 A_3)A_4))$, $((A_1 A_2)(A_3 A_4))$, $((A_1(A_2 A_3))A_4)$, $(((A_1 A_2) A_3)A_4)$

Matrix-chain multiplication problem

Fully parenthesize the product $A_1 A_2 \cdots A_n$, where A_i is $p_{i-1} \times p_i$, in a way that minimizes the total number of scalar multiplications.

Brute force search: the number of distinct ways is

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k),\ P(1) = 1 \Rightarrow P(n) = c_{n-1},$$

where $c_n = \dfrac{1}{n+1}\dbinom{2n}{n} \sim \dfrac{4^n}{\sqrt{\pi} n^{3/2}}$ is the

n-th ***Catalan number***. Inefficient!

Optimal substructure

Consider the subproduct

$$A_{i..j} = A_i A_{i+1} \cdots A_j\ ,\ 1 \le i \le j \le n.$$

- If $i = j$, then optimum cost is 0.
- If $i < j$, we cut the chain somewhere in the middle: choose a k, $i \le k < j$, then compute $A_{i..k}$, $A_{k+1..j}$, and their product. If we already know the optimal cost for $A_{i..k}$ and $A_{k+1..j}$ *for all* k, then the optimum cost for $A_{i..j}$ can be found by minimizing the total cost over all k.

Recursive structure

Let $m[i, j]$ be the optimum cost for $A_{i..j}$.

- If $i = j$, then $m[i, j] = 0$.
- If $i < j$, then

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

Let $s[i, j]$ be the smallest minimizing value of k.

Natural recursive solution: running time satisfies

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

$$T(n) \ge 2^{n-1}$$

Few overlapping subproblems

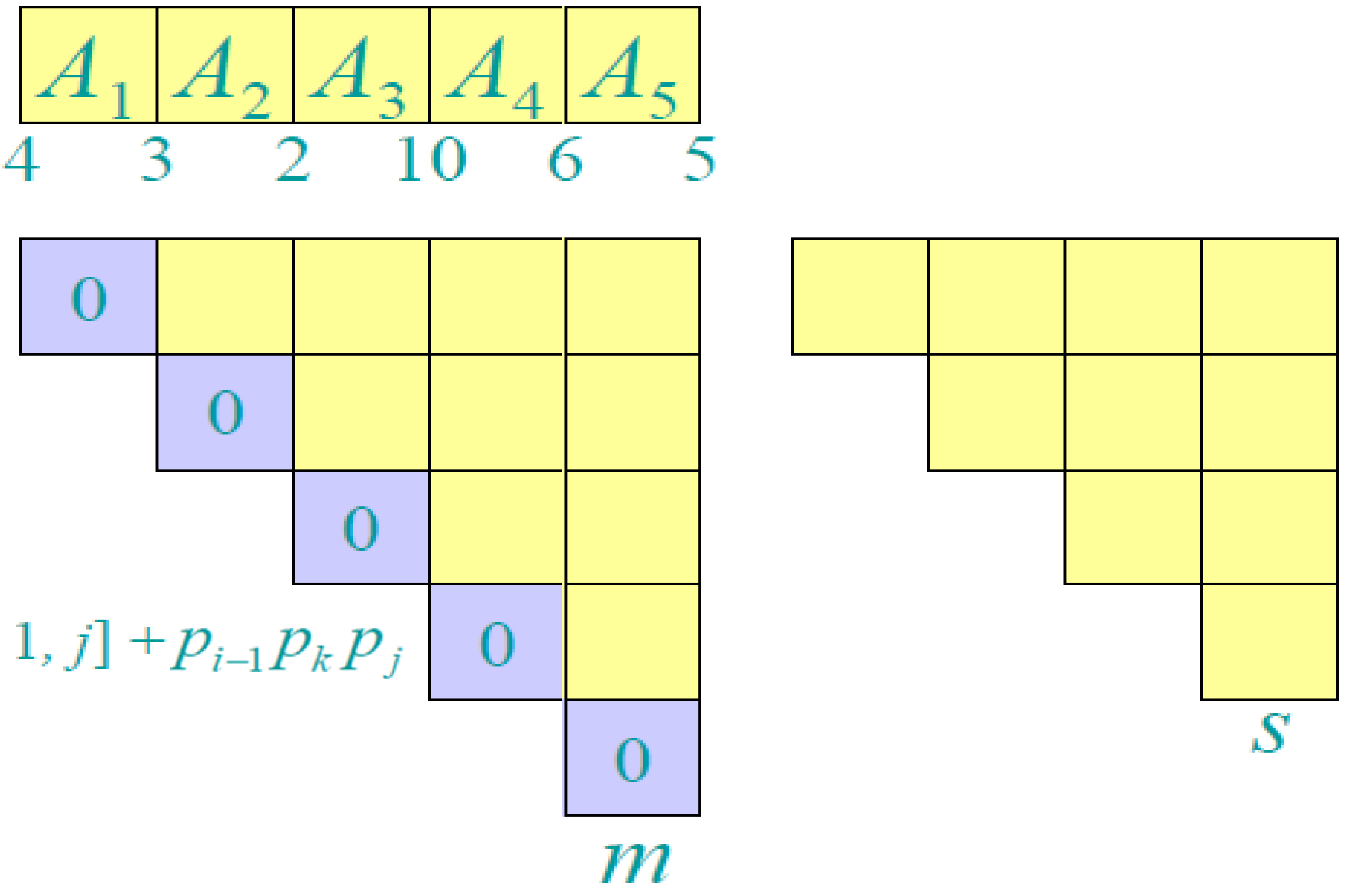
There are only as many subproblems as there are pairs (i, j) such that $1 \le i \le j \le n$, that is, $n(n+1)/2$.

These subproblems overlap: $m[i, j]$ is referred to during the computation of $m[i', j']$ whenever the interval $[i, j]$ is properly contained in $[i', j']$.

We can thus use dynamic programming to compute the optimum costs bottom-up.

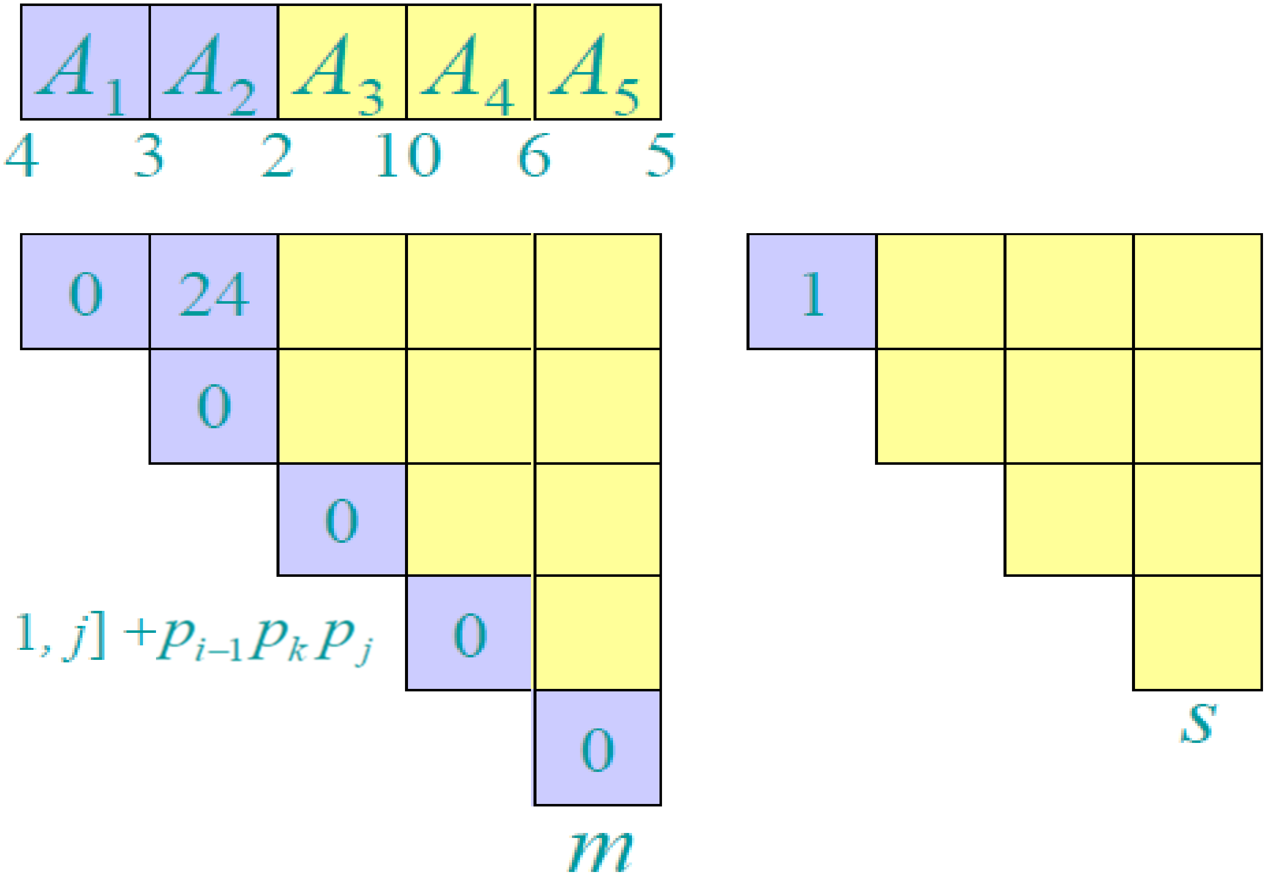
Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i–1} p_k p_j
        if q < m[i, j] then
          m[i, j] ← q
          s[i, j] ← k
  return m and s
```



Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i–1} p_k p_j
        if q < m[i, j] then
          m[i, j] ← q
          s[i, j] ← k
  return m and s
```



Matrix chain order

MATRIX-CHAIN-ORDER (p)

```
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, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j] then
                m[i, j] ← q
                s[i, j] ← k
return m and s
```

A_1	A_2	A_3	A_4	A_5

4 3 2 10 6 5

0	24			
	0	60		
		0		
			0	
				0

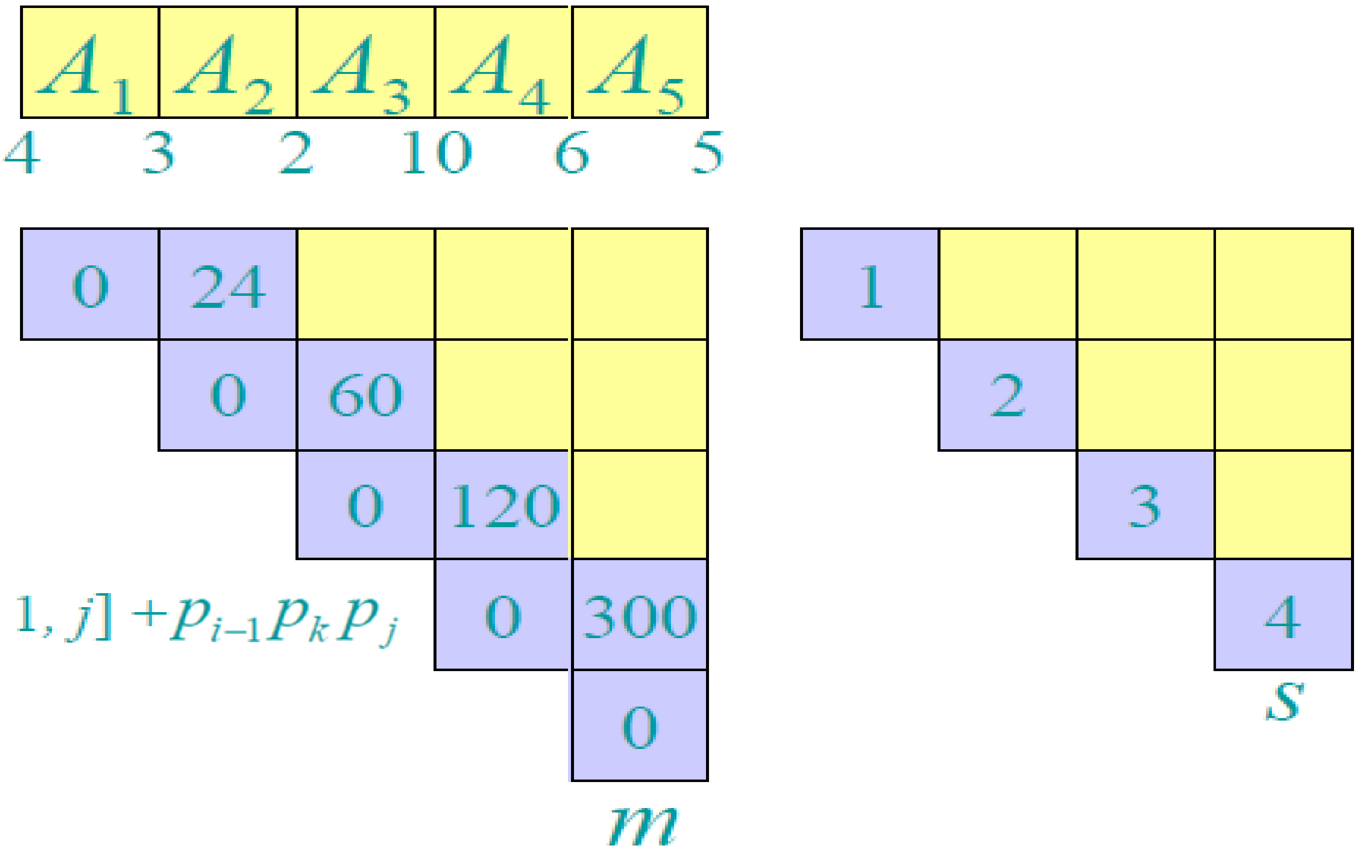
m

1			
	2		

s

Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i−1} p_k p_j
        if q < m[i, j] then
          m[i, j] ← q
          s[i, j] ← k
  return m and s
```



Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i−1} p_k p_j
           if q < m[i, j] then
              m[i, j] ← q
              s[i, j] ← k
  return m and s
```

A_1	A_2	A_3	A_4	A_5

4 3 2 10 6 5

m

0	24	104		
	0	60		
		0	120	
			0	300
				0

s

1	2		
	2		
		3	
			4

$$m[1,3]=\min\{0+60+4\cdot 3\cdot 10, 24+0+4\cdot 2\cdot 10\}.$$

Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i−1} p_k p_j
        if q < m[i, j] then
          m[i, j] ← q
          s[i, j] ← k
  return m and s
```

A_1	A_2	A_3	A_4	A_5

4 3 2 10 6 5

0	24	104	192	
	0	60	156	
		0	120	180
			0	300
				0

m

1	2	2	
	2	2	
		3	4
			4

s

$$m[1,4]=\min\{0+156+4\cdot3\cdot6,\ 24+120+4\cdot2\cdot6,\ 104+0+4\cdot10\cdot6\}.$$

Matrix chain order

```
MATRIX-CHAIN-ORDER (p)
  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, k] + m[k + 1, j] + p_{i–1} p_k p_j
        if q < m[i, j] then
          m[i, j] ← q
          s[i, j] ← k
  return m and s
```

A_1	A_2	A_3	A_4	A_5

4 3 2 10 6 5

0	24	104	192	244
	0	60	156	210
		0	120	180
			0	300
				0

m

1	2	2	2
	2	2	2
		3	4
			4

s

Running time: $\Theta(n^3)$ Space requirement: $\Theta(n^2)$.

Matrix chain order

Optimal solution

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

```
PRINT-OPTIMAL-PARENS (s, i, j)
  if i = j then
     print "A"_i
  else
    print "("
    PRINT-OPTIMAL-PARENS (s, i, s[i, j])
    PRINT-OPTIMAL-PARENS (s, s[i, j]+1, j)
    print ")"
```

Implementation

Matrix chain order

MATRIX-CHAIN-ORDER (p)
$n \leftarrow length[p] - 1$
for $i \leftarrow 1$ **to** n **do**
  $m[i, i] \leftarrow 0$
for $l \leftarrow 2$ **to** n **do**
  for $i \leftarrow 1$ **to** $n - l + 1$ **do**
    $j \leftarrow i + l - 1$
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i$ **to** $j - 1$ **do**
      $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$
      if $q < m[i, j]$ **then**
        $m[i, j] \leftarrow q$
        $s[i, j] \leftarrow k$
return m and s

A_1	A_2	A_3	A_4	A_5

4 3 2 10 6 5

0	24	104	192	244
	0	60	156	210
		0	120	180
			0	300
				0

m

1	2	2	2
	2	2	2
		3	4
			4

s

Running time: $\Theta(n^3)$ Space requirement: $\Theta(n^2)$.

```python
import sys

# Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
def MatrixChainOrder(p, n):

  # for simplicity access
  m = [[0 for x in range(n)] for x in range(n)]

  # initialization
  for i in range(1, n):
    m[i][i] = 0

  # L is chain length.
  for L in range(2, n):
    for i in range(1, n-L + 1):
      j = i + L-1
      m[i][j] = sys.maxsize
      for k in range(i, j):

        # scalar multiplications
        q = m[i][k] + m[k + 1][j] + p[i-1]*p[k]*p[j]
        if q < m[i][j]:
          m[i][j] = q

  return m[1][n-1]

# Program input
arr = [4, 3, 2, 10, 6, 5]
size = len(arr)

print("Minimum number of multiplications is " +
  str(MatrixChainOrder(arr, size)))
```

```
Minimum number of multiplications is 244
```

Example code test

- Code test: https://www.acmicpc.net/problem/11049
- Solving the problem using dynamic programming

THANK YOU