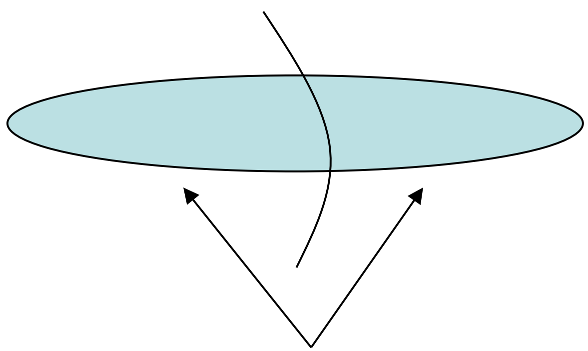
Design & Analysis of Algorithms CSE 304

Dynamic Programming

DP - Two key ingredients

- Two key ingredients for an optimization problem to be suitable for a dynamic-programming solution:

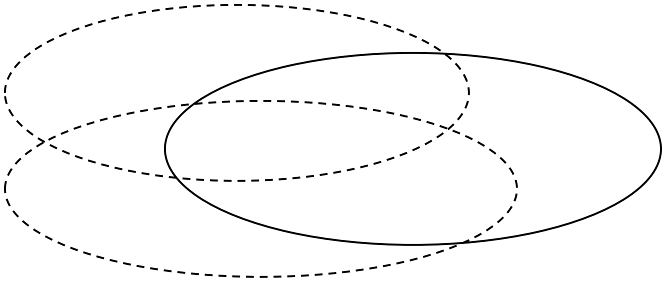
1. optimal substructures



Each substructure is optimal.

(Principle of optimality)

2. overlapping subproblems



Subproblems are dependent.

(otherwise, a divide-and-conquer approach is the choice.)

Matrix-chain Multiplication

- Suppose we have a sequence or chain A_1, A_2, …, A_n of n matrices to be multiplied
  - That is, we want to compute the product $A_1A_2\ldots A_n$

- There are many possible ways (parenthesizations) to compute the product

Matrix-chain Multiplication …contd

- Example: consider the chain A_1, A_2, A_3, A_4 of 4 (i.e. *n*) matrices
  - Let us compute the product $A_1A_2A_3A_4$

$$P(n) = P(n-1) + P(2) \times P(n-2) + P(3) \times P(n-3) + \cdots + P(n-1) \times P(1)$$

```
def P(n):
    if n == 1 or n == 2:
        return 1
    total = 0
    for i in range(1, n):
        total += P(i) * P(n-i)
    return total

P_4 = P(4)
P_5 = P(5)
P_8 = P(8)
P_10 = P(10)
print(P_4, P_5, P_8, P_10)
```

```
5 14 429 4862
```

- There are 5 possible ways:
  1. $(A_1(A_2(A_3A_4)))$
  2. $(A_1((A_2A_3)A_4))$
  3. $((A_1A_2)(A_3A_4))$
  4. $((A_1(A_2A_3))A_4)$
  5. $(((A_1A_2)A_3)A_4)$

Matrix-chain Multiplication ...contd

- To compute the number of scalar multiplications necessary, we must know:
  - Algorithm to multiply two matrices
  - Matrix dimensions

- Algorithm to multiply two matrices?

Algorithm to Multiply 2 Matrices

Input: Matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p\times q$ and $q\times r$)

Result: Matrix $C_{p\times r}$ resulting from the product $A \cdot B$

MATRIX-MULTIPLY($A_{p\times q}$, $B_{q\times r}$)

1. **for** $i \leftarrow 1$ **to** p
2. **for** $j \leftarrow 1$ **to** r
3. $C[i, j] \leftarrow 0$
4. **for** $k \leftarrow 1$ **to** q
5. $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
6. **return** C

Scalar multiplication in line 5 dominates time to compute C
Number of scalar multiplications = pqr

Matrix-chain Multiplication ...contd

- Example: Consider three matrices $A_{10\times100}$, $B_{100\times5}$, and $C_{5\times50}$
- There are 2 ways to parenthesize
  - $((AB)C) = D_{10\times5} \cdot C_{5\times50}$
    - $AB \Rightarrow 10\cdot100\cdot5=5{,}000$ scalar multiplications
    - $DC \Rightarrow 10\cdot5\cdot50 =2{,}500$ scalar multiplications

    Total: 7,500
  - $(A(BC)) = A_{10\times100} \cdot E_{100\times50}$
    - $BC \Rightarrow 100\cdot5\cdot50=25{,}000$ scalar multiplications
    - $AE \Rightarrow 10\cdot100\cdot50 =50{,}000$ scalar multiplications

    Total: 75,000

Matrix-Chain Multiplication

- Given a chain of matrices $\langle A_1, A_2, \ldots, A_n \rangle$, where for i = 1, 2, …, n matrix A_i has dimensions $p_{i-1} x\, p_i$, fully parenthesize the product $A_1 \cdot A_2 \cdot\cdot\cdot A_n$ in a way that minimizes the number of scalar multiplications.

$$A_1 \quad \cdot \quad A_2 \quad \cdot\cdot\cdot \quad A_i \quad \cdot \quad A_{i+1} \quad \cdot\cdot\cdot \quad A_n$$

$$p_0 \times p_1 \quad p_1 \times p_2 \quad p_{i-1} \times p_i \quad p_i \times p_{i+1} \quad p_{n-1} \times p_n$$

2. A Recursive Solution

- Consider the subproblem of parenthesizing

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

  $p_{i-1}p_kp_j$

  $m[i, k]$ $= A_{i\ldots k}\, A_{k+1\ldots j}$ $m[k+1,j]$ for $i \le k < j$

- Assume that the optimal parenthesization splits the product $A_i\, A_{i+1} \cdots A_j$ at k ($i \le k < j$)

$$m[i, j] = \underbrace{m[i, k]}_{} + \underbrace{m[k+1, j]}_{} + \underbrace{p_{i-1}p_kp_j}_{}$$

min # of multiplications to compute $A_{i\ldots k}$

min # of multiplications to compute $A_{k+1\ldots j}$

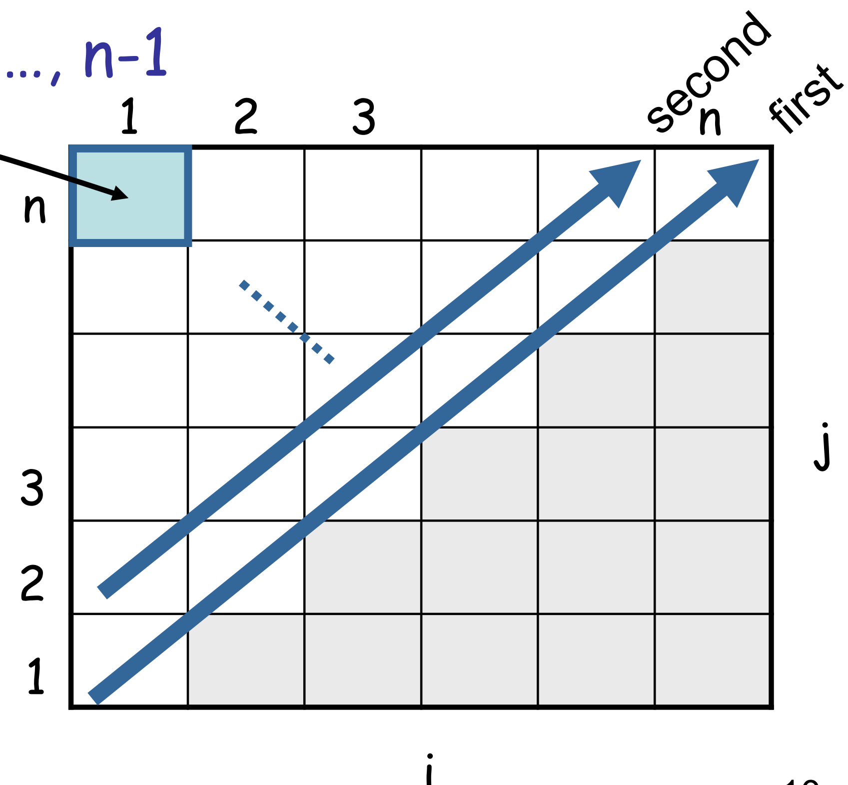
# of multiplications to compute $A_{i\ldots k}A_{k\ldots j}$

3. Computing the Optimal Costs

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

- Length = 1: $i = j$, $i = 1, 2, \ldots, n$
- Length = 2: $j = i + 1$, $i = 1, 2, \ldots, n-1$

m[1, n] gives the optimal solution to the problem

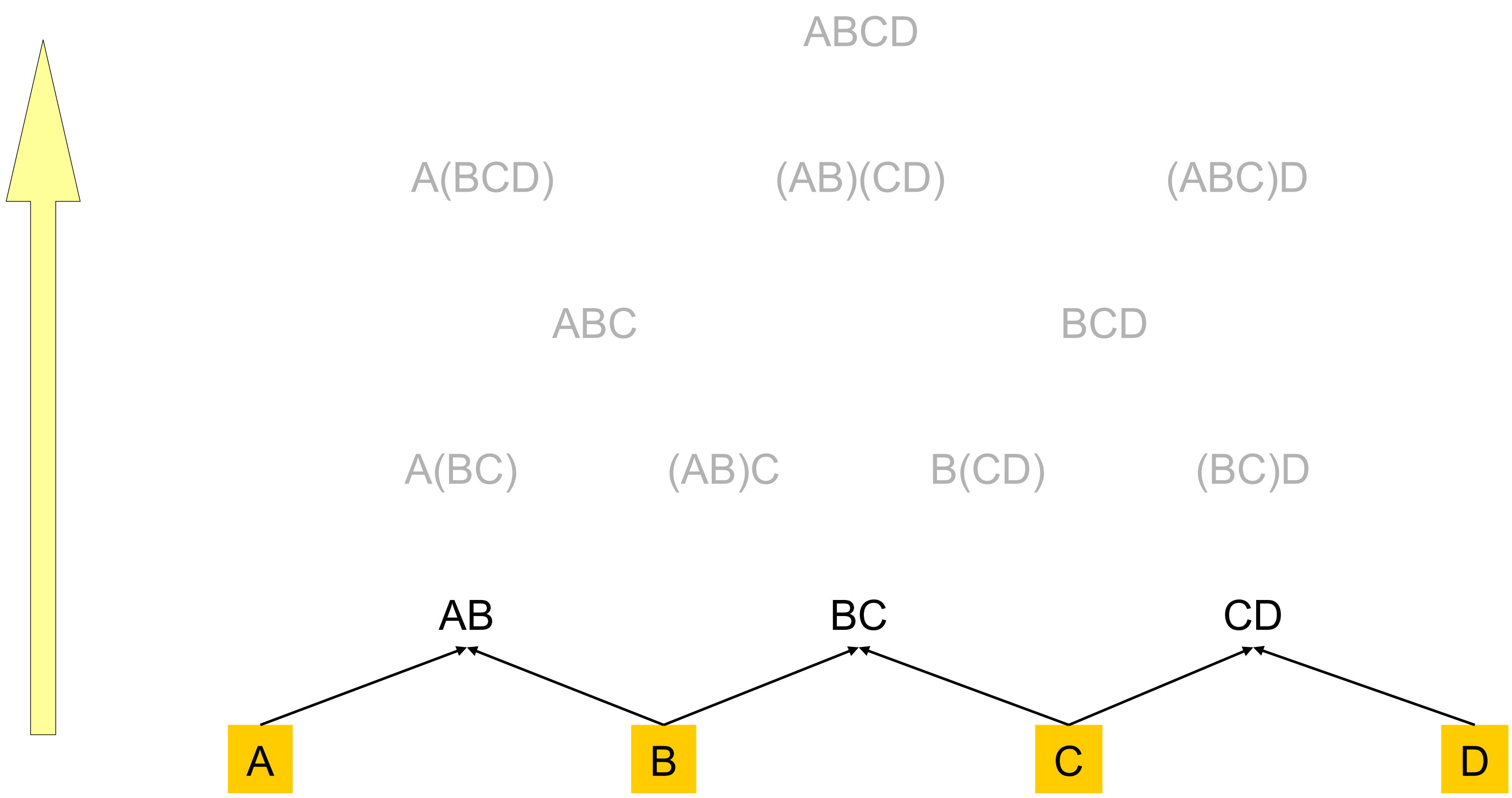


Compute rows from bottom to top and from left to right

In a similar matrix s we keep the optimal values of k

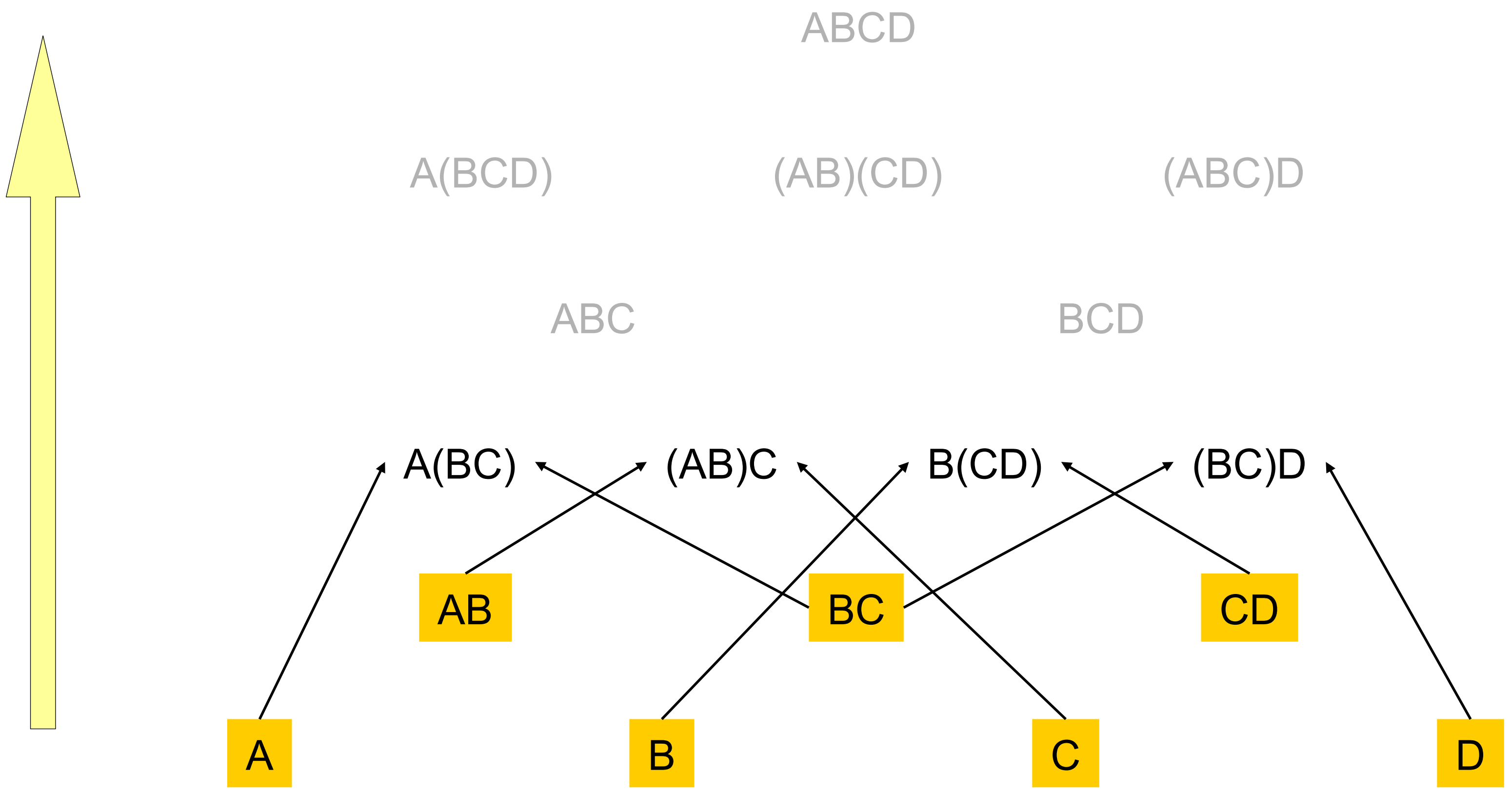
Multiply 4 Matrices: A×B×C×D (1)

- Compute the costs in the bottom-up manner
  - First we consider AB, BC, CD
  - No need to consider AC or BD



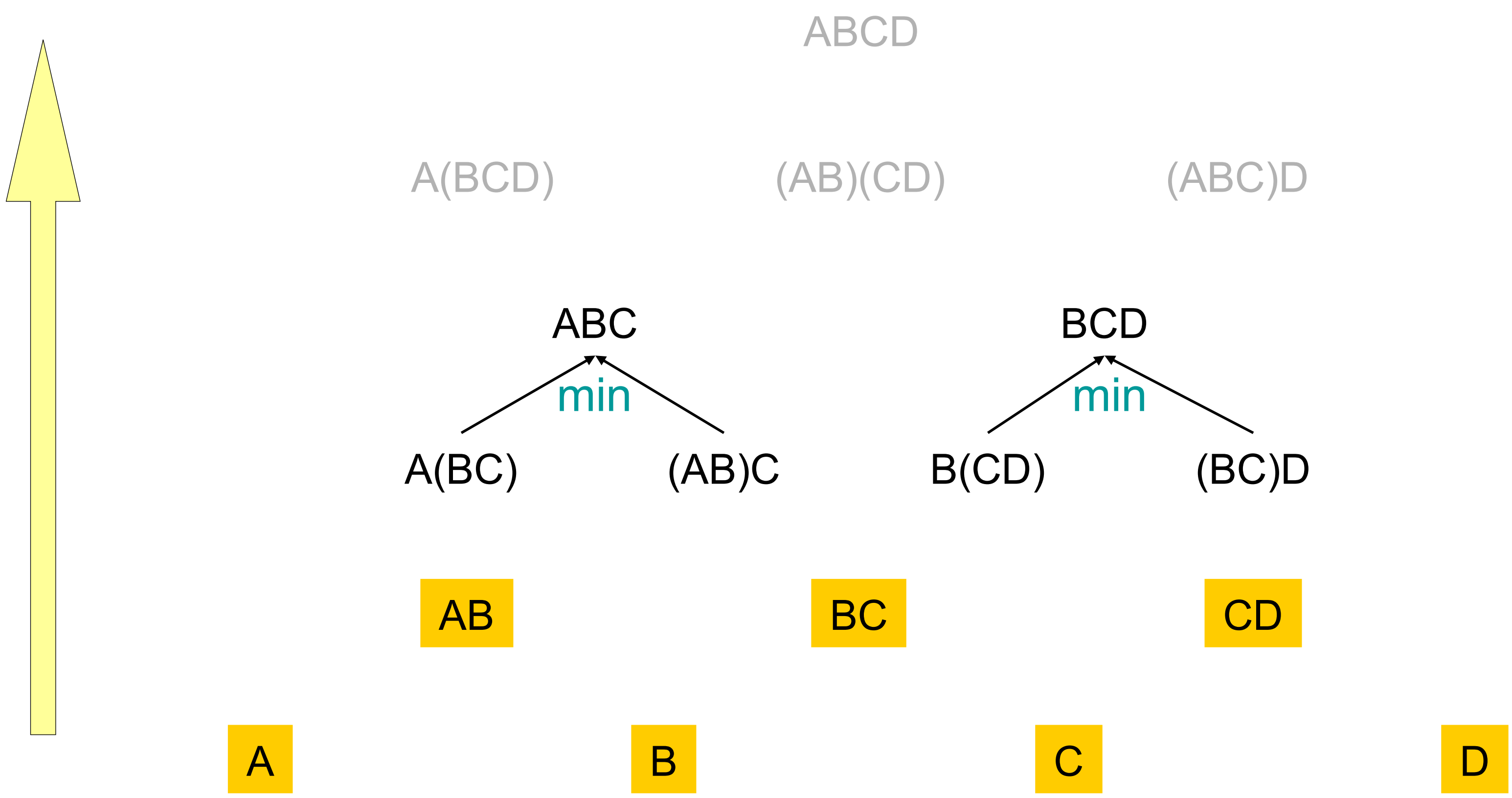
Multiply 4 Matrices: A×B×C×D (2)

- Compute the costs in the bottom-up manner
  - Then we consider A(BC), (AB)C, B(CD), (BC)D
  - No need to consider (AB)D, A(CD)



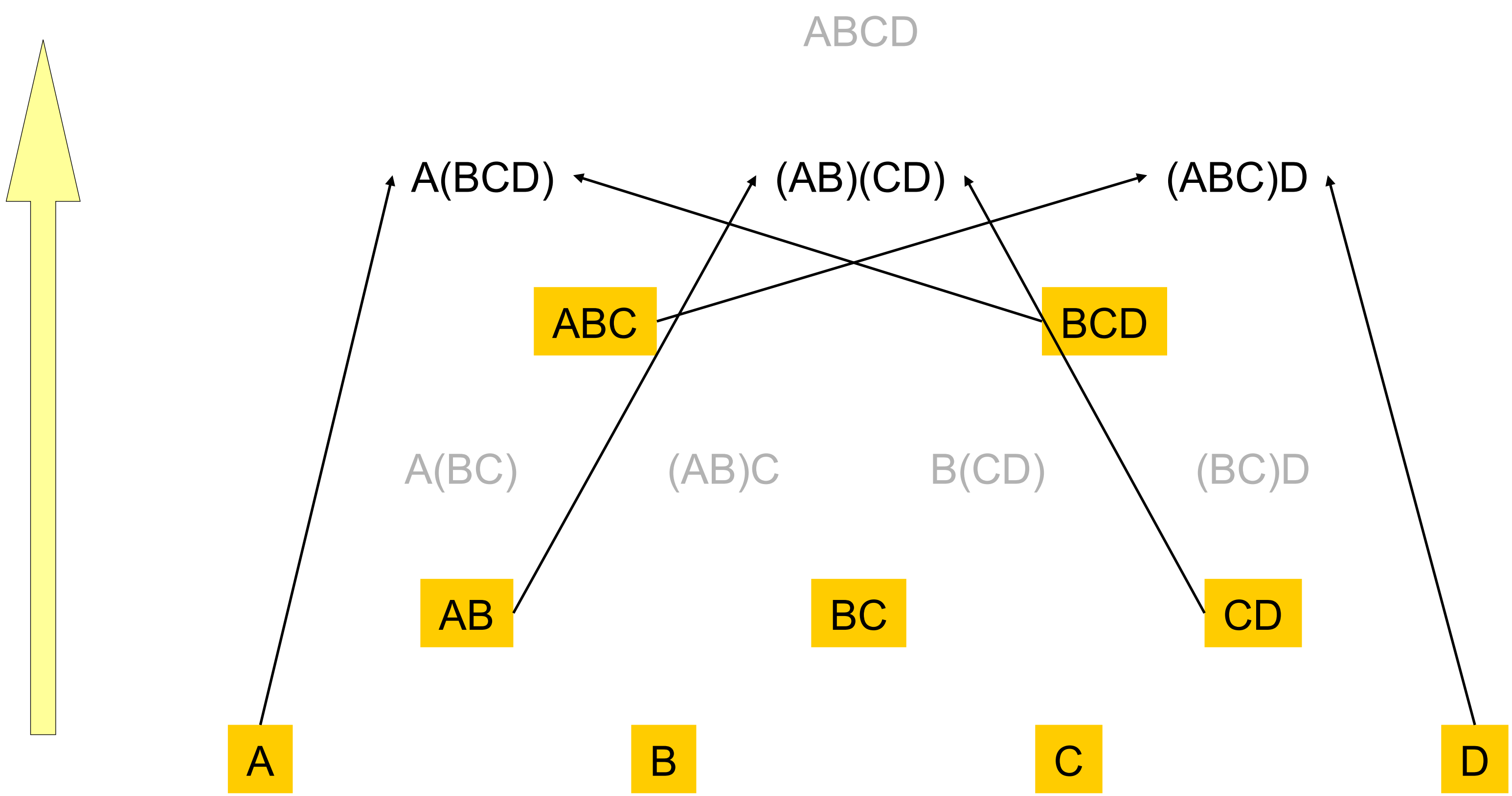
Multiply 4 Matrices: A×B×C×D (3)

- Compute the costs in the bottom-up manner
- Select minimum cost matrix calculations of ABC & BCD



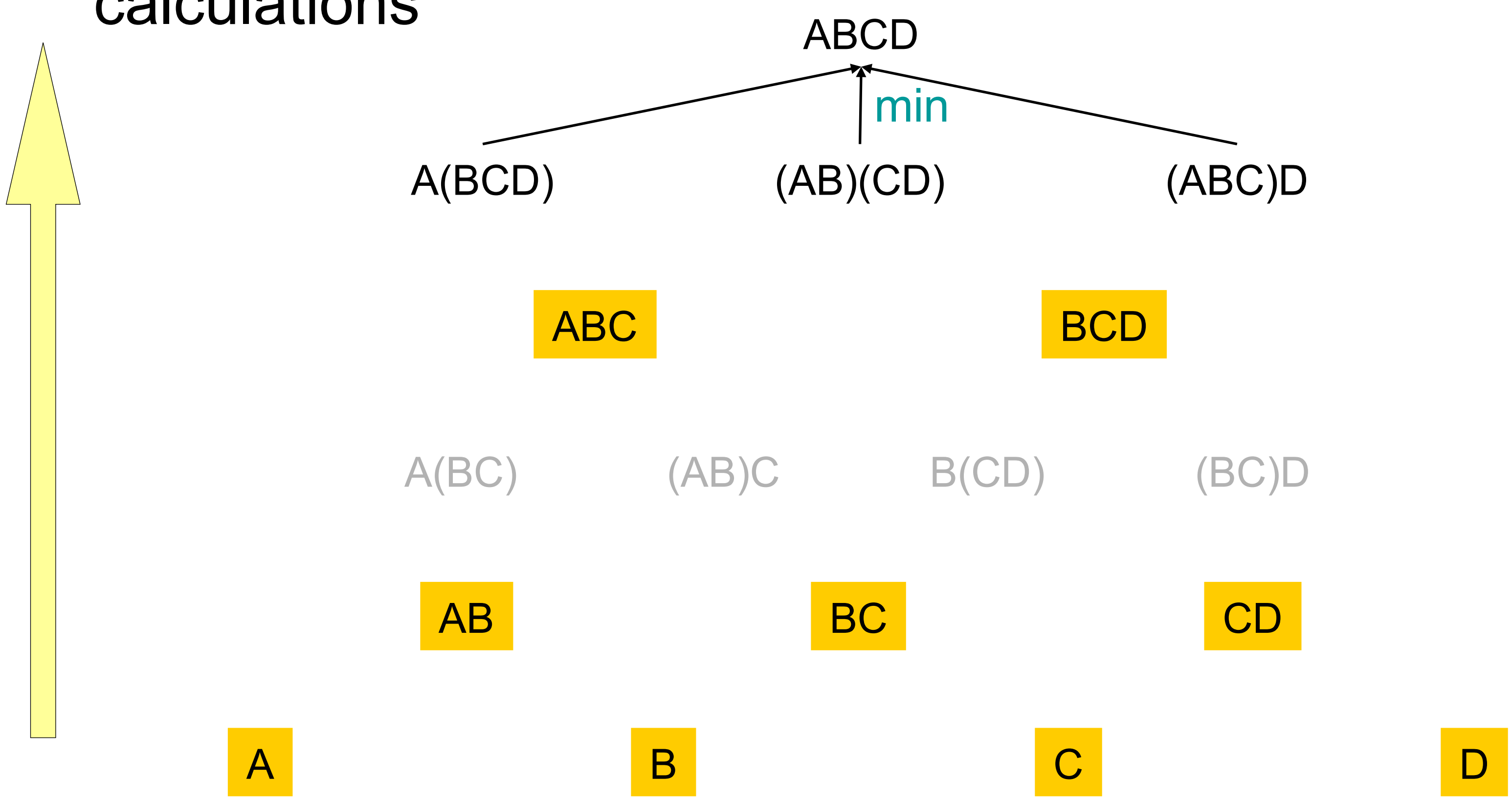
Multiply 4 Matrices: A×B×C×D (4)

- Compute the costs in the bottom-up manner
  - We now consider A(BCD), (AB)(CD), (ABC)D



Multiply 4 Matrices: A×B×C×D (5)

- Compute the costs in the bottom-up manner
  - Finally choose the cheapest cost plan for matrix calculations



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

	1	2	3	4
1	0			
2		0		
3			0	
4				0

m(i,j), i ≤ j

	1	2	3	4
1				
2				
3				
4				

k(i,j), i ≤ j

$A_1 \times A_2 \times A_3 \times A_4$

4 3   3 5   5 2   2 6

p_0 p_1 p_2 p_3 p_4

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

	1	2	3	4
1	0	60		
2		0	30	
3			0	60
4				0

m(i,j), i ≤ j

	1	2	3	4
1		1		
2			2	
3				3
4				

k(i,j), i ≤ j

$$A_1 \times A_2 \times A_3 \times A_4$$

4 3 | 3 5 | 5 2 | 2 6

p_0 p_1 p_2 p_3 p_4

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

	1	2	3	4
1	0	60	54	
2		0	30	
3			0	60
4				0

m(i,j), i ≤ j

	1	2	3	4
1		1	1	
2			2	
3				3
4				

k(i,j), i ≤ j

A_1	×	A_2	×	A_3	×	A_4
4 3		3 5		5 2		2 6

p_0 p_1 p_2 p_3 p_4

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

	1	2	3	4
1	0	60	54	
2		0	30	66
3			0	60
4				0

m(i,j), i ≤ j

	1	2	3	4
1		1	1	
2			2	3
3				3
4				

k(i,j), i ≤ j

$A_1 \times A_2 \times A_3 \times A_4$

4 3 3 5 5 2 2 6

p_0 p_1 p_2 p_3 p_4

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

	1	2	3	4
1	0	60	54	102
2		0	30	66
3			0	60
4				0

m(i,j), i ≤ j

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

k(i,j), i ≤ j

$A_1 \times A_2 \times A_3 \times A_4$

A_1		A_2		A_3		A_4	
4	3	3	5	5	2	2	6

p_0 p_1 p_2 p_3 p_4

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

$A_1 \times A_2 \times A_3 \times A_4$

4 3 3 5 5 2 2 6

p_0 p_1 p_2 p_3 p_4

$A_1 \times A_2 \times A_3 \times A_4$

4 3 3 5 5 2 2 6

p_0 p_1 p_2 p_3 p_4

	1	2	3	4
1	0	60	54	102
2		0	30	66
3			0	60
4				0

m(i,j), i ≤ j

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

k(i,j), i ≤ j

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

	1	2	3	4
1	0			
2		0		
3			0	
4				0

m(i,j), i ≤ j

	1	2	3	4
1				
2				
3				
4				

k(i,j), i ≤ j

$$A_1 \times A_2 \times A_3 \times A_4$$

A_1		A_2		A_3		A_4	
4	3	3	5	5	2	2	6

p_0 p_1 p_2 p_3 p_4

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

	1	2	3	4
1	0			
2		0		
3			0	
4				0

m(i,j), i ≤ j

	1	2	3	4
1				
2				
3				
4				

k(i,j), i ≤ j

$A_1 \times A_2 \times A_3 \times A_4$

4 3 3 5 5 2 2 6

p_0 p_1 p_2 p_3 p_4

MATRIX-CHAIN-ORDER(p)

```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i,j] = ∞
        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
return m and s
```

PRINT-OPTIMAL-PARENS(s, i, j)

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

Memoization

- Top-down approach with the efficiency of typical dynamic programming approach
- Maintaining an entry in a table for the solution to each subproblem
  - **memoize** the inefficient recursive algorithm
- When a subproblem is first encountered its solution is computed and stored in that table
- Subsequent “calls” to the subproblem simply look up that value

Memoized Matrix-Chain

Alg.: MEMOIZED-MATRIX-CHAIN(p)

1. $n \leftarrow$ length[p] - 1
2. **for** $i \leftarrow 1$ **to** n
3. **do for** $j \leftarrow i$ **to** n
4. **do** $m[i, j] \leftarrow \infty$
5. **return** LOOKUP-CHAIN(p, 1, n)

Initialize the m table with large values that indicate whether the values of $m[i, j]$ have been computed

Top-down approach

Memoized Matrix-Chain

Alg.: LOOKUP-CHAIN(p, i, j)

Running time is $O(n^3)$

if $m[i, j] < \infty$
  then return $m[i, j]$
if $i = j$
  then $m[i, j] \leftarrow 0$
  else for $k \leftarrow i$ **to** $j - 1$
    do $q \leftarrow$ LOOKUP-CHAIN(p, i, k) + LOOKUP-CHAIN(p, k+1, j) + $p_{i-1}p_kp_j$
    if $q < m[i, j]$
      then $m[i, j] \leftarrow q$

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

Dynamic Progamming vs. Memoization

- Advantages of dynamic programming vs. memoized algorithms
  - No overhead for recursion, less overhead for maintaining the table
  - The regular pattern of table accesses may be used to reduce time or space requirements
- Advantages of memoized algorithms vs. dynamic programming
  - Some subproblems do not need to be solved