Dynamic Programming

MATRIX CHAIN MULTIPLICATION

Matrix Chain Multiplication Problem

Multiplying non-square matrices:

- A is $p \times q$, B is $q \times r$
- AB is $p \times r$ whose (i, j) entry is $\sum a_{ik}\, b_{kj}$

Must be equal

Computing AB takes $p \cdot q \cdot r$ scalar multiplications and $p(q-1)r$ scalar additions (using basic algorithm).

Suppose we have a sequence of matrices to multiply. What is the best order?

Matrix Chain Multiplication Problem

Given a sequence of matrices $A_1, A_2, \ldots, A_n$, then

Compute $C = A_1 . A_2 . \ldots, A_n$

Different ways to compute C

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

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

- Matrix multiplication is associative
  - So output will be the same
- However, time cost can be very different
  - Example

Why Order Matters

Suppose we have 4 matrices:

- *A*, 30 × 01
- *B*, 01 × 40
- *C*, 40 × 10
- *D*, 10 × 25

((*AB*)(*CD*)) : requires 41,200 multiplications

[(30×1×40) + (40×10×25) +(30×40×25) = 41,200]

(*A*((*BC*)*D*)) : requires 1400 multiplications

[(1×40×10) + (1×10×25) +(30×1×25) = 1,400]

Matrix Chain Multiplication Problem

Given a sequence of matrices $A_1, A_2, \ldots, A_n$, where A_i is $p_{i-1} \times p_i$:

- What is minimum number of scalar multiplications required to compute $A_1 \cdot A_2 \cdot \ldots \cdot A_n$?
- What order of matrix multiplications achieves this minimum?

- Fully parenthesize the product in a way that minimizes the number of scalar multiplications

(( )( ))(( )(( )(( )( ))))

No. of parenthesizations: ???

A Possible Solution

Try all possibilities and choose the best one.

Drawback is there are too many of them (exponential in the number of matrices to be multiplied)

The number of parenthesizations is

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

The solution to the recurrence is $\Omega(2^n)$

No. of parenthesizations: Exponential

Need to be more clever – try dynamic programming.

Step 1: Optimal Substructure Property

A problem exhibits *optimal substructure* if an optimal solution to the problem contains within it optimal solutions to subproblems.

Whenever a problem exhibits optimal substructure, we have a good clue that dynamic programming might apply.

Consequently, we must take care to ensure that the range of subproblems we consider includes those used in an optimal solution.

We must also take care to ensure that the total number of distinct subproblems is a polynomial in the input size.

Step 1: Optimal Substructure Property

Define $A_{i..j}$, $i \le j$, to be the matrix that results from evaluating the product $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$.

If the problem is nontrivial, i.e, $i < j$, then to parenthesize $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$, split the product between A_k and A_{k+1} for some k, where $i \le k < j$.

- The cost of parenthesizing this way is
  - The cost of computing the matrix $A_{i..k}$ +
  - The cost of computing the matrix $A_{k+1..j}$ +
  - The cost of multiplying them together
- The optimal substructure of this problem is:

  An optimal parenthesization of $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$ contains within it optimal parenthesizations of $A_i \cdot A_{i+1} \cdot \ldots \cdot A_k$ and $A_{k+1} \cdot A_{k+2} \cdot \ldots \cdot A_j$

  Proof ?

Overlapping Subproblem Property

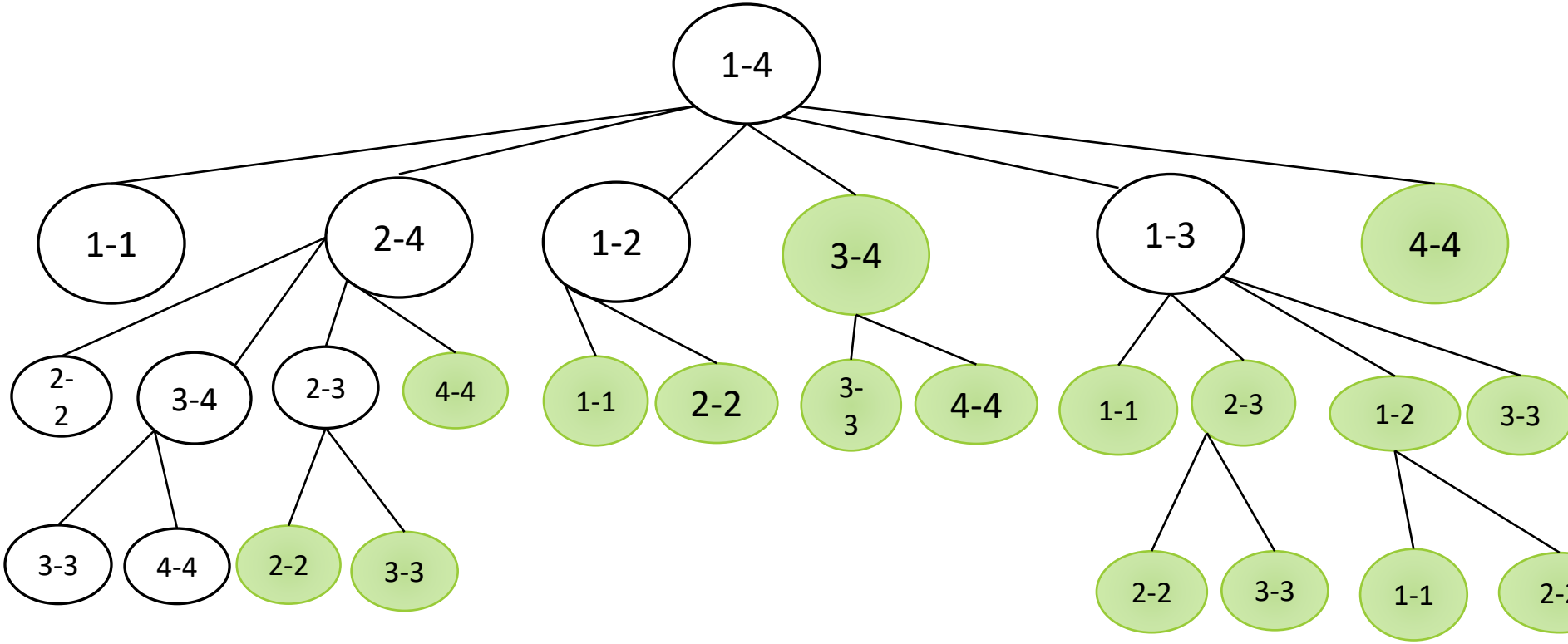
Two subproblems of the same problem are independent if they do not share resources.

Two subproblems are *overlapping* if they are really the same subproblem that occurs as a subproblem of different problems.

A problem exhibits *overlapping subproblem* if the number of subproblems is “small” in the sense that a recursive algorithm solves the same subproblems over and over, rather than always generating new subproblems.

Dynamic programming algorithms take advantage of overlapping subproblems by solving each subproblem once and then storing the solution in a table where it can be looked up when needed.

Overlapping Subproblem Property



Matrix Chain Multiplication

Suppose You have 04 Matrices

A_1 A_2 A_3 A_4

5x4 4x6 6x2 2x7

m represents cost of multiplication

$A_1 \times A_1$ m[1,1]= 0;

$A_2 \times A_2$ m[2,2]= 0;

$A_3 \times A_3$ m[3,3]= 0;

$A_4 \times A_4$ m[4,4]= 0;

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

s	1	2	3	4
1				
2				
3				
4				

Matrix Chain Multiplication

Suppose You have 04 Matrices

A_1 A_2 A_3 A_4

5x4 4x6 6x2 2x7

m represents cost of multiplication

$A_1 \times A_2$

m[1,2]= 5x6x4 = 120;

$A_2 \times A_3$

m[2,3]= 4x2x6= 48;

$A_3 \times A_4$

m[3,4]= 6x7x2= 84;

m	1	2	3	4
1	0	120		
2		0	48	
3			0	84
4				0

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

Matrix Chain Multiplication

Suppose You have 04 Matrices

A_1 A_2 A_3 A_4

5x4 4x6 6x2 2x7

m represents cost of multiplication

$A_1 \times A_2 \times A_3$

$A_1 (A_2.A_3)$

$m[1,1] + m[2,3] + d_0 d_1 d_3$

$= 0 + 48 + 5x4x2 = 88;$

$(A_1.A_2).A_3$

$m[1,2] + m[3,3] + d_0 d_2 d_3$

$= 120 + 0 + 5x6x2 = 180;$

m	1	2	3	4
1	0	120	88	
2		0	48	
3			0	84
4				0

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

Matrix Chain Multiplication

Suppose You have 04 Matrices

A_1 A_2 A_3 A_4

5x4 4x6 6x2 2x7

m represents cost of multiplication

$A_2 \times A_3 \times A_4$

$A_2 (A_3.A_4)$

$m[2,2] + m[3,4] + d_1d_2d_4$

$= 0 + 84 + 4x6x7 = 252;$

$(A_2.A_3).A_4$

$m[2,3] + m[4,4] + d_1d_3d_4$

$= 48 + 0 + 4x2x7 = 104;$

m	1	2	3	4
1	0	120	88	
2		0	48	104
3			0	84
4				0

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

Matrix Chain Multiplication

Suppose You have 04 Matrices

A_1 A_2 A_3 A_4

5x4 4x6 6x2 2x7

m represents cost of multiplication

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

$A_1 (A_2 A_3.A_4)$

$m[1,1] + m[2,4] + d_0 d_1 d_4$

$= 0 + 104 + 5x4x7 = 244;$

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

$m[1,2] + m[3,4] + d_0 d_2 d_4$

$= 120 + 84 + 5x6x7 = 414;$

$(A_1. A_2.A_3).A_4$

$m[1,3] + m[4,4] + d_0 d_3 d_4$

$= 88 + 0 + 5x2x7 = 158;$

m	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

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

Now The Formula

- Define $m[i, j]$ to be the minimum number of multiplications needed to compute $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$.
  - Goal: Find $m[1, n]$
  - Basis: $m[i, i] = 0$
  - Recursion: How to define $m[i, j]$ recursively ?

- Consider all possible ways to split A_i through A_j into two pieces.
- Compare the costs of all these splits:
  - best case cost for computing the product of the two pieces
  - plus the cost of multiplying the two products
- Take the best one

Now The Formula

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

Step 2: Develop a Recursive Solution

```
Matrix_Chain_RecursiveWay (m, i, j)
1.      for (i <- 0 to n-1)
2.          for (j <-i to n-1)
                    m[ i, j] <- ∞
1.              if(i=j)
2.                  then return 0;
3.              if(m[ i, j] ≠ ∞)
4.                  then return m[i, j];
5.              for (k<- i to j)
6.                  do cost= Matrix_Chain_RecursiveWay (m, i, k) +
                            Matrix_Chain_RecursiveWay (m, k+1, j) + d_{i-1} d_k d_j;
7.                  if(cost < m[ i, j])
8.                          then m[ i , j] <- cost;
9.              return m [ i, j];
```

Memorization

running time
$O(n^3)$

Step 2: Develop a Recursive Solution

- Let $T(n)$ be the time taken by Recursive-Matrix-Chain for n matrices.

$$T(1) \geq 1$$

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

For $i = 1, 2, \ldots, n-1$, each term $T(i)$ appears once as $T(k)$ and once as $T(n-k)$. Thus, we have

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

$$\geq 2^{n-1}$$

Then $T(n) = \Omega(2^n)$

Step 3: Compute the Optimal Costs

Find Dependencies among Subproblems

m:

	1	2	3	4	5
1	0				
2	n/a	0			
3	n/a	n/a	0		
4	n/a	n/a	n/a	0	
5	n/a	n/a	n/a	n/a	0

GOAL

computing the red square requires the blue ones: to the left and below.

Step 3: Compute the Optimal Costs

Find Dependencies among Subproblems

m:

	1	2	3	4	5
1	0				
2	n/a	0			
3	n/a	n/a	0		
4	n/a	n/a	n/a	0	
5	n/a	n/a	n/a	n/a	0

- Computing $m(i, j)$ uses
  - everything in same row to the left:

    $m(i, i)$, $m(i, i+1)$, …, $m(i, j-1)$
  - and everything in same column below:

    $m(i+1, j)$, $m(i+2, j)$, …, $m(j, j)$

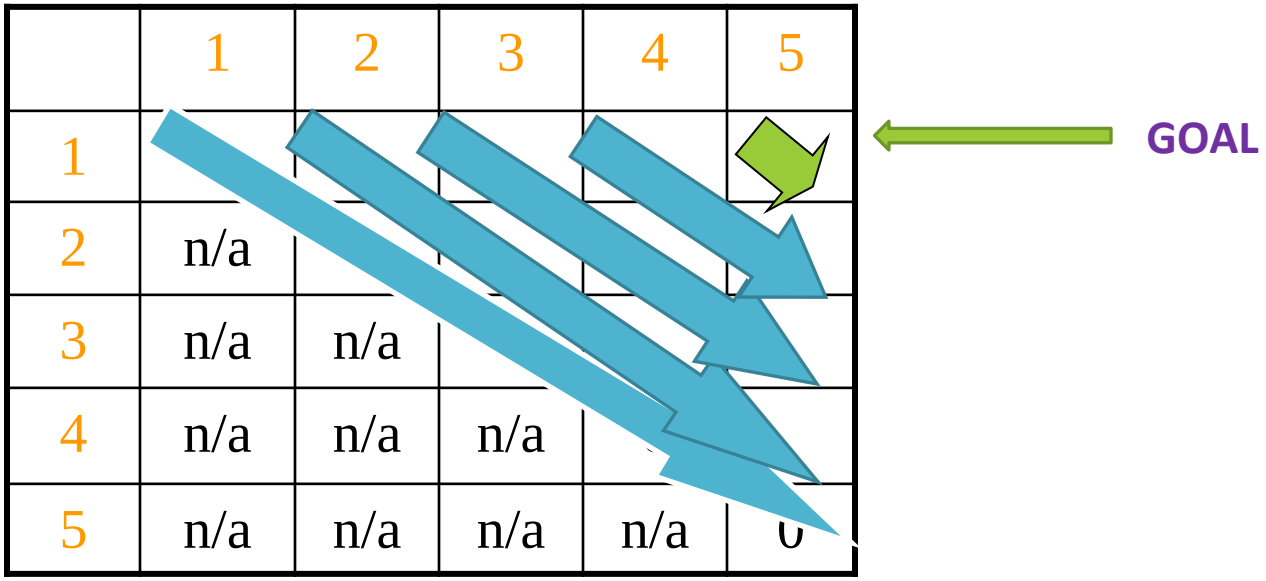
Step 3: Compute the Optimal Costs

Identify Order for Solving Subproblems

- Solve the subproblems (i.e., fill in the table entries) this way:
  - go along the diagonal
  - start just above the main diagonal
  - end in the upper right corner (goal)

m:

	1	2	3	4	5
1					
2	n/a				
3	n/a	n/a			
4	n/a	n/a	n/a		
5	n/a	n/a	n/a	n/a	0



Step 4: Construct an Optimal Solution

- It's fine to know the cost of the cheapest order, but what is that cheapest order?
- Keep another array *s* and update it when computing the minimum cost in the inner loop
- After *m* and *s* have been filled in, then call a recursive algorithm on *s* to print out the actual order

```
Print_order(s, i, j)
1.    if (i=j)
          then print "A"i;
      Else
          print "("
          Print_order (s, i, s[i,j] );
          Print_order (s, s[i,j] +1, j);
          print ")";
```

THANK YOU!