Today:
- Matrix Chain Multiplication

COSC 581, Algorithms

January 23, 2014

Reading Assignments

- Today's class:
  - Chapter 15.2

- Reading assignment for next class:
  - Chapter 15.3-15.4

Matrix Chain Multiplication

- Given some matrices to multiply, determine the ***best*** order to multiply them so you minimize the number of single element multiplications.
  - i.e., Determine the way the matrices are fully parenthesized.

- First, it should be noted that matrix multiplication is associative, but not commutative. But since it is associative, we always have:

- ((AB)(CD)) = (A(B(CD))), or any other grouping as long as the matrices are in the same consecutive order.

- BUT NOT: ((AB)(CD)) = ((BA)(DC))

Matrix Chain Multiplication

- It may appear that the amount of work done won't change if you change the parenthesization of the expression, but we can prove that is not the case!

- FIRST, remember some matrix multiplication rules…

  To multiply matrix A, which is size p x q

  with matrix B, which is size q x r

  The resulting matrix is of what size?

Matrix Chain Multiplication

- It may appear that the amount of work done won't change if you change the parenthesization of the expression, but we can prove that is not the case!

- FIRST, remember some matrix multiplication rules…

  To multiply matrix A, which is size p x q

  with matrix B, which is size q x r

  The resulting matrix is of what size? p x r

  Number of scalar multiplications needed?

Matrix Chain Multiplication

- It may appear that the amount of work done won't change if you change the parenthesization of the expression, but we can prove that is not the case!

- FIRST, remember some matrix multiplication rules…

To multiply matrix A, which is size p x q

with matrix B, which is size q x r

The resulting matrix is of what size? p x r

Number of scalar multiplications needed? p x q x r

Matrix Chain Multiplication

- Let us examine the following example:
  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix

- We will show that the way we group matrices when multiplying A, B, C *can greatly affect number of required scalar multiplications*:

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC)**:
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400.

- Consider computing **(AB)C**:
  - # multiplications for (AB) =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400.

- Consider computing **(AB)C**:
  - # multiplications for (AB) = 2x10x50 = 1000, creating a 2x50 answer matrix
  - # multiplications for (AB)C =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400.

- Consider computing **(AB)C**:
  - # multiplications for (AB) = 2x10x50 = 1000, creating a 2x50 answer matrix
  - # multiplications for (AB)C = 2x50x20 = 2000,
  - Total multiplications =

Matrix Chain Multiplication

- Let A be a 2x10 matrix
- Let B be a 10x50 matrix
- Let C be a 50x20 matrix

- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400.

- Consider computing **(AB)C**:
  - # multiplications for (AB) = 2x10x50 = 1000, creating a 2x50 answer matrix
  - # multiplications for (AB)C = 2x50x20 = 2000
  - Total multiplications = 1000 + 2000 = 3000

Matrix Chain Multiplication

- Thus, our goal today is:
- Given a chain of matrices to multiply, determine the fewest number of scalar multiplications necessary to compute the product.
- Note: we don't actually need to compute the multiplication – just the ordering of the multiplications

How Many Possible Parenthesizations?

- For $n \geq 2$, a fully parenthesized matrix product is the product of 2 fully parenthesized matrix subproducts.
- The split can occur between k^{th} and $(k+1)^{th}$ matrices, for any k = 1, 2, …, n-1
- So, the recurrence representing the total # of possible parenthesizations is:
  - $P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$
- Solution is tricky -- turns out, it grows as $\Omega\left(\frac{4^n}{n^{3/2}}\right)$
- Or, also true that it grows as $\Omega(2^n)$

Matrix Chain Multiplication

- Formal Definition of the problem:
  - Let $A = A_1 \bullet A_2 \bullet \ldots A_n$
  - And let $p_{i-1} \times p_i$ denote the dimensions of matrix A_i.
  - We must find the minimal number of scalar multiplications necessary to calculate A
    - assuming that each single matrix multiplication uses the simple “standard” (9^{th} grade ☺) method.

Recall:
The Primary Steps of Dynamic Programming

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.

Step 1: Optimal Substructure

- The key to solving this problem is noticing the ***optimal substructure***:
  - If a particular parenthesization of the whole product is optimal, then any sub-parenthesization in that product is optimal as well.

- *Or, stating the same thing through an example:*
  - ***If*** (A (B ((CD) (EF)))) is optimal
  - Then (B ((CD) (EF))) is optimal as well
  - ***Illustration of Proof on the next slide…***

Optimal Substructure

- Assume that we are calculating ABCDEF and that the following parenthesization is optimal:

  (A (B ((CD) (EF))))

  Then it is necessarily the case that

  (B ((CD) (EF)))

  is the optimal parenthesization of BCDEF.

- Why is this?
  - Because if it weren't, and, say, (((BC) (DE)) F) were better, then it would also follow that

    (A (((BC) (DE)) F)) was better than

    (A (B ((CD) (EF)))),
  - contradicting its optimality!

Optimal Substructure

Optimal

$$((A_1 A_2 \ldots A_k)(A_{k+1} A_{k+2} \ldots A_n))$$

Combine

Matrix Chain Multiplication

- Our final multiplication will ALWAYS be of the form

  $(A_1 \bullet A_2 \bullet \ldots A_k) \bullet (A_{k+1} \bullet A_{k+2} \bullet \ldots A_n)$

- In essence, there is exactly one value of k for which we should "split" our work into two separate cases so that we get an optimal result.
  - Here is a list of the cases to choose from:
  - $(A_1) \bullet (A_2 \bullet A_3 \bullet \ldots A_n)$
  - $(A_1 \bullet A_2) \bullet (A_3 \bullet A_4 \bullet \ldots A_n)$
  - $(A_1 \bullet A_2 \bullet A_3) \bullet (A_4 \bullet A_5 \bullet \ldots A_n)$
  - ...
  - $(A_1 \bullet A_2 \bullet \ldots A_{n-2}) \bullet (A_{n-1} \bullet A_n)$
  - $(A_1 \bullet A_2 \bullet \quad \ldots \quad A_{n-1}) \bullet (A_n)$

- Basically, count the number of multiplications in each of these choices and **pick the minimum**.
  - One other point to notice is that you have to account for the minimum number of multiplications in each of the two products.

Matrix Chain Multiplication

- Consider the case multiplying these 4 matrices:
  - A: 2x4
  - B: 4x2
  - C: 2x3
  - D: 3x1
- 1. (A)(BCD) - This is a 2x4 multiplied by a 4x1,
  - so 2x4x1 = 8 multiplications, plus whatever work it will take to multiply (BCD).
- 2. (AB)(CD) - This is a 2x2 multiplied by a 2x1,
  - so 2x2x1 = 4 multiplications, plus whatever work it will take to multiply (AB) and (CD).
- 3. (ABC)(D) - This is a 2x3 multiplied by a 3x1,
  - so 2x3x1 = 6 multiplications, plus whatever work it will take to multiply (ABC).

Step 2: A recursive solution

- Define *m*[*i, j*] = minimum number of scalar multiplications needed to compute the matrix $A_{i..j} = A_i A_{i+1} \dots A_j$
- Goal *m*[*1, n*] (i.e., $A_{1..n} = A_1 A_2 \dots A_n$)

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

- Since *m*[*i,j*] only gives value of optimal solution, we also define *s*[*i,j*] to be a value of *k* at which we split the product $A_{i..j} = A_i A_{i+2} \dots A_j$ in an optimal parenthesization

Step 3: Computing the Optimal Costs

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

- Now let’s turn this recursive formula into a dynamic programming solution
  - Which sub-problems are necessary to solve first?
  - Clearly it's necessary to solve the smaller problems before the larger ones.
  - Here “smaller” means shorter matrix chains
  - So, solve for matrix chains of length 1, then of length 2, …

Step 3:
Computing the optimal costs

```
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
```

Here, we're checking different places to "split" our matrices by checking different values of k and seeing if they improve our current minimum value.

Step 3: Computing the optimal costs

```
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
```

Here, we’re checking different places to “split” our matrices by checking different values of k and seeing if they improve our current minimum value.

Complexity?

Step 3:
Computing the optimal costs

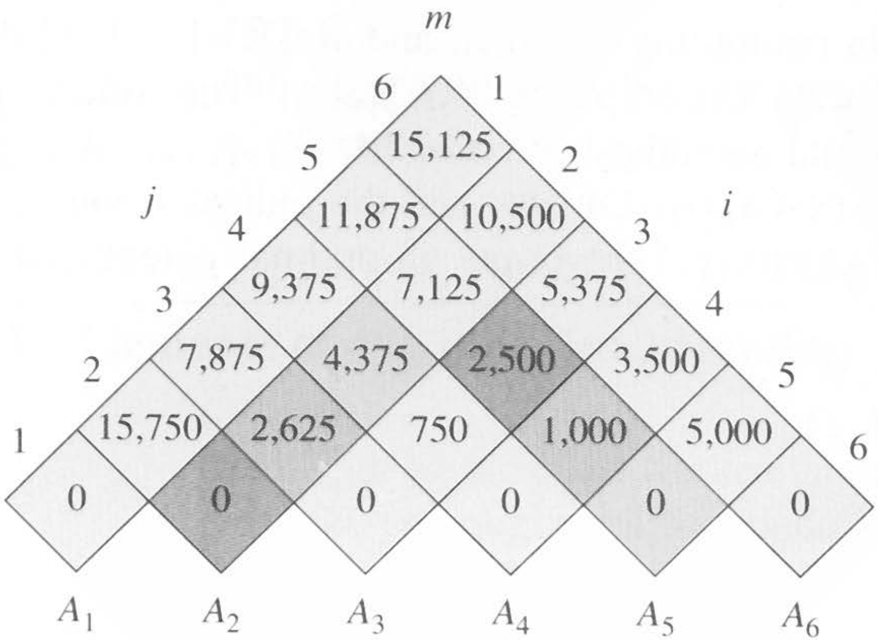
MATRIX-CHAIN-ORDER(p)

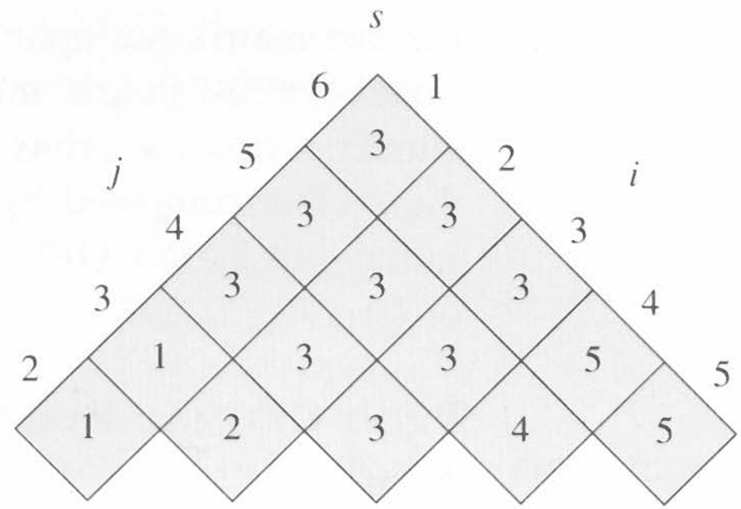
```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                      // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Here, we're checking different places to "split" our matrices by checking different values of k and seeing if they improve our current minimum value.

Complexity? $\Theta(n^3)$

Example m and s tables computed by MATRIX-CHAIN-ORDER for n=6



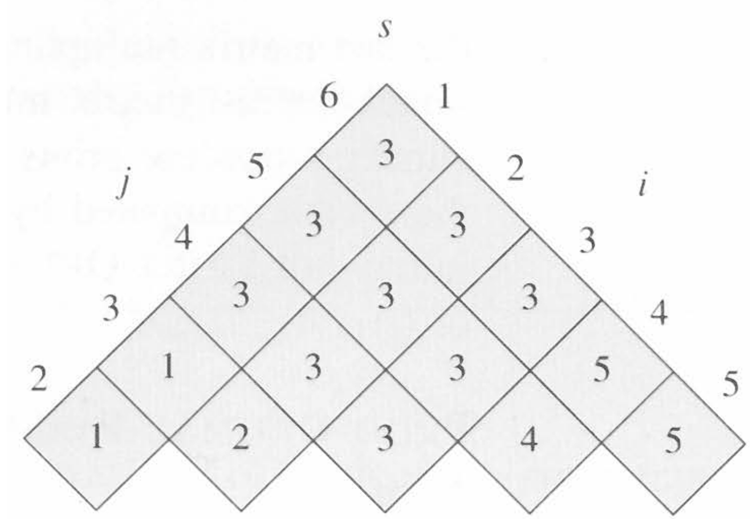


Step 4:
Constructing an optimal solution

```
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 ")"
```

Step 4:
Constructing an optimal solution

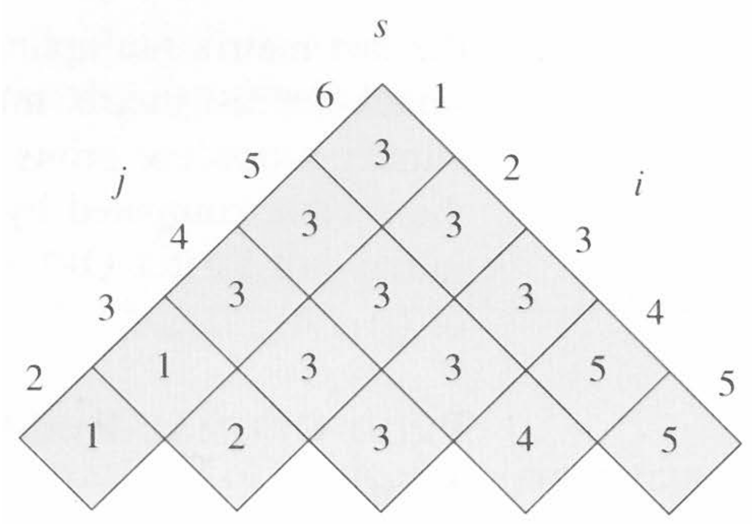
```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
      print "A"_i
else print "("
      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
      print ")"
```



Example: $A_1 \cdots A_6$

Step 4: Constructing an optimal solution

```
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 ")"
```



Example: $A_1 \cdots A_6$

$$(A_1(A_2A_3))\big((A_4A_5)(A_6)\big)$$

In-Class Exercise

- Describe a dynamic programming algorithm to find the maximum product of a contiguous sequence of positive numbers A[1..n].

- For example, if A = (0.1, 17, 1, 5, 0.5, 0.2, 4, 0.7, 0.02, 12, 0.3), then the answer would be 85 because of the subsequence (17, 1, 5)

Reading Assignments

- Today's class:
  - Chapter 15.2

- Reading assignment for next class:
  - Chapter 15.3-15.4