CS4311
Design and Analysis of Algorithms

Lecture 10: Dynamic Programming II
Matrix Multiplication

• Let $A$ be a matrix of dimension $p \times q$ and $B$ be a matrix of dimension $q \times r$.

• Then, if we multiply matrices $A$ and $B$, we obtain a resulting matrix $C = AB$ whose dimension is $p \times r$.

• We can obtain each entry in $C$ using $q$ operations $\Rightarrow$ in total, $pqr$ operations.
Matrix Multiplication

• Let $A$ be a matrix of dimension $p \times q$
  and $B$ be a matrix of dimension $q \times r$

• Then, if we multiply matrices $A$ and $B$, we obtain a resulting matrix $C = AB$
  whose dimension is $p \times r$

• We can obtain each entry in $C$ using $q$ operations $\Rightarrow$ in total, $pqr$ operations
Matrix Multiplication

Example:

\[
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} \\
a_{2,1} & a_{2,2} & a_{2,3}
\end{pmatrix}
\begin{pmatrix}
b_{1,1} & b_{1,2} \\
b_{2,1} & b_{2,2} \\
b_{3,1} & b_{3,2}
\end{pmatrix}
= 
\begin{pmatrix}
c_{1,1} & c_{1,2} \\
c_{2,1} & c_{2,2}
\end{pmatrix}
\]

How to obtain \( c_{1,2} \)?
Matrix Multiplication

• In fact, \(((A_1A_2)A_3) = (A_1(A_2A_3))\) so that matrix multiplication is associative

⇒ Any way to write down the parentheses gives the same result

E.g., \(((((A_1A_2)A_3)A_4) = (((A_1A_2)(A_3A_4))

  = (A_1((A_2A_3)A_4)) = (((A_1(A_2A_3))A_4)

  = (A_1(A_2(A_3A_4))))
Matrix Multiplication

Question: Why do we bother this?

Because different computation sequence may use different number of operations!
E.g., Let the dimensions of $A_1, A_2, A_3$ be:

\[ \begin{align*}
1 \times 100, & \quad 100 \times 1, \quad 1 \times 100, \quad \text{respectively} \\
\end{align*} \]

#operations to get $(A_1A_2)A_3) = ??$

#operations to get $(A_1(A_2A_3)) = ??$
Lemma: Suppose that to multiply $B_1, B_2, \ldots, B_j$, the way with minimum #operations is to:

(i) first, obtain $B_1B_2 \ldots B_x$
(ii) then, obtain $B_{x+1} \ldots B_j$
(iii) finally, multiply the matrices of part (i) and part (ii)

Then, the matrices in part (i) and part (ii) must be obtained with min #operations.
Optimal Substructure

Let \( f_{i,j} \) denote the min #operations to obtain the product \( A_i A_{i+1} \ldots A_j \)

\[ f_{i,i} = 0 \]

Let \( r_k \) and \( c_k \) denote #rows and #cols of \( A_k \)

Then, we have:

**Lemma:** For any \( j > i \),

\[ f_{i,j} = \min_x \{ f_{i,x} + f_{x+1,j} + r_i c_x c_j \} \]
Define a function $\text{Compute}_F(i, j)$ as follows:

$$\text{Compute}_F(i, j) \quad /* \text{Finding } f_{i,j} */$$

1. if ($i == j$) return 0;
2. $m = \infty$;
3. for ($x = i, i+1, ..., j-1$) {
   
   $g = \text{Compute}_F(i, x) + \text{Compute}_F(x+1,j) + r_i c_x c_j$;
   
   if ($g < m$) $m = g$;

4. return $m$;
Matrix-Chain Multiplication

Question: Time to get \( \text{Compute}_F(1,n) \)?

- By substituion method, we can show that
  \[
  \text{Running time} = \Omega(3^n)
  \]

- Remark: On the other hand, \#operations for each possible way of writing parentheses are computed at most once \( \Rightarrow \) Running time = \( O( \frac{C(2n-2,n-1)}{n} ) \)

\[\text{Catalan Number}\]
Overlapping Subproblems

Here, we can see that:

To Compute\(_F(i,j)\) and Compute\(_F(i,j+1)\), both have many COMMON subproblems:
Compute\(_F(i,i+1)\), ..., Compute\(_F(i,j-1)\)

So, in our recursive algorithm, there are many redundant computations!

Question: Can we avoid it?
Bottom-Up Approach

• We notice that
  \[ f_{i,j} \text{ depends only on } f_{x,y} \text{ with } |x-y| < |i-j| \]

• Let us create a 2D table \( F \) to store all \( f_{i,j} \) values once they are computed

• Then, compute \( f_{i,j} \) for \( j-i = 1,2,\ldots,n-1 \)
Bottom-Up Approach

```c
BottomUp_F() /* Finding min #operations */
1. for j = 1, 2, ..., n, set F[j, j] = 0;
2. for (length = 1, 2, ..., n-1) {
    Compute F[i, i+length] for all i;
    // Based on F[x, y] with |x-y| < length
}
3. return F[1, n];
```

Running Time = $\Theta(n^3)$
Remarks

• Again, a slight change in the algorithm allows us to get the exact sequence of steps (or the parentheses) that achieves the minimum number of operations.

• Also, we can make minor changes to the recursive algorithm and obtain a memoized version (whose running time is $O(n^3)$).