# CS4311
# Design and Analysis of Algorithms

## Lecture 10: Dynamic Programming II

# About this lecture

- We will see more examples today

# 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 → in total, $pqr$ operations

# Matrix Multiplication

- Matrix multiplication can be defined for more than two matrices
- Let $A_1, A_2, \ldots, A_n$ be a sequence of matrices (with #columns of $A_k$ = # rows of $A_{k+1}$)
- We can define

$$C = A_1 A_2 A_3 \ldots A_n = (\ldots((A_1 A_2) A_3) \ldots A_n)$$

# 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:

$1 \times 100$, $100 \times 1$, $1 \times 100$ , respectively

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

#operations to get $(A_1(A_2A_3))$ = ??

# Optimal Substructure

Lemma: Suppose that to multiply $B_1, B_2, \ldots, B_j$, the way with minimum #operations is to:

(i) first, obtain $B_1 B_2 \ldots B_x$

(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 $j > i$,

$$f_{i,j} = \min_x \{ f_{i,x} + f_{x+1,j} + r_i c_x c_j \}$$

# Matrix-Chain Multiplication

Define a function Compute_F(i,j) as follows:

Compute_F(i, j) /* Finding $f_{i,j}$ */

1. if (i == j) return 0;
2. m = ∞;
3. for (x = i, i+1, …, j-1) {

   g = Compute_F(i,x) + 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 Compute_F(1,n)?

- By substituion method, we can show that

  Running time = $\Omega(3^n)$
- On the other hand, #operations for each possible way of writing parentheses are computed at most once

  ➔ Running time = $O( C(2n-2,n-1)/n )$

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

  (i) all $f_{i,j}$ are eventually computed at least once, and

  (ii) $f_{i,j}$ depends only on $f_{x,y}$ with $|x-y|<|i-j|$

- By (i), let us create a 2D table $F$ to store all $f_{i,j}$ values once they are computed
- By (ii), let us compute $f_{i,j}$ for $j-i = 1,2,\ldots,n-1$

# Bottom-Up Approach

```
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 $O(n^3)$)