

**NPTEL MOOC**

# **PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON**

**Week 8, Lecture 4**

**Madhavan Mukund, Chennai Mathematical Institute**

**<http://www.cmi.ac.in/~madhavan>**



# Multiplying matrices

- \* To multiply matrices A and B, need compatible dimensions
  - \* A of dimension  $m \times n$ , B of dimension  $n \times p$
  - \* AB has dimension  $mp$
- \* Each entry in AB take  $O(n)$  steps to compute
  - \*  $AB[i,j]$  is  $A[i,1]B[1,j] + A[i,2]B[2,j] + \dots + A[i,n]B[n,j]$
- \* Overall, computing AB is  $O(mnp)$



# Multiplying matrices

- \* Matrix multiplication is associative
- \*  $ABC = (AB)C = A(BC)$
- \* Bracketing does not change the answer ...
- \* ... but can affect the complexity of computing it!



# Multiplying matrices

- \* Suppose dimensions are  $A[1,100]$ ,  $B[100,1]$ ,  $C[1,100]$
- \* Computing  $A(BC)$ 
  - \*  $BC$  is  $[100,100]$ ,  $100 \times 1 \times 100 = 10000$  steps
  - \*  $A(BC)$  is  $[1,100]$ ,  $1 \times 100 \times 100 = 10000$  steps
- \* Computing  $(AB)C$ 
  - \*  $AB$  is  $[1,1]$ ,  $1 \times 100 \times 1 = 100$  steps
  - \*  $(AB)C$  is  $[1,100]$ ,  $1 \times 1 \times 100 = 100$  steps
- \*  $A(BC)$  takes 20000 steps,  $(AB)C$  takes 200 steps!



# Multiplying matrices

- \* Given matrices  $M_1, M_2, \dots, M_n$  of dimensions  $[r_1, c_1], [r_2, c_2], \dots, [r_n, c_n]$
- \* Dimensions match, so  $M_1 \times M_2 \times \dots \times M_n$  can be computed
- \*  $c_i = r_{i+1}$  for  $1 \leq i < n$
- \* Find an optimal order to compute the product
  - \* That is, bracket the expression optimally



# Inductive structure

- \* Product to be computed:  $M_1 \times M_2 \times \dots \times M_n$
- \* Final step would have combined two subproducts
  - \*  $(M_1 \times M_2 \times \dots \times M_k) \times (M_{k+1} \times M_{k+2} \times \dots \times M_n)$ , for some  $1 \leq k < n$
  - \* First factor has dimension  $(r_1, c_k)$ , second  $(r_{k+1}, c_n)$
  - \* Final multiplication step costs  $O(r_1 c_k c_n)$
  - \* Add cost of computing the two factors



# Subproblems

- \* Final step is  $(M_1 \times M_2 \times \dots \times M_k) \times (M_{k+1} \times M_{k+2} \times \dots \times M_n)$
- \* Subproblems are  $(M_1 \times M_2 \times \dots \times M_k)$  and  $(M_{k+1} \times M_{k+2} \times \dots \times M_n)$
- \* Total cost is  $\text{Cost}(M_1 \times M_2 \times \dots \times M_k) + \text{Cost}(M_{k+1} \times M_{k+2} \times \dots \times M_n) + r_1 C_k C_n$
- \* Which  $k$  should we choose?
- \* No idea! Try them all and choose the minimum!



# Inductive formulation

- \*  $\text{Cost}(M_1 \times M_2 \times \dots \times M_n) =$   
minimum value, for  $1 \leq k < n$ , of  
 $\text{Cost}(M_1 \times M_2 \times \dots \times M_k) +$   
 $\text{Cost}(M_{k+1} \times M_{k+2} \times \dots \times M_n) +$   
 $r_1 C_k C_n$
- \* When we compute  $\text{Cost}(M_1 \times M_2 \times \dots \times M_k)$  we will  
get subproblems of the form  $M_j \times M_{j+1} \times \dots \times M_k$



# In general ...

- \*  $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_j) =$   
minimum value, for  $i \leq k < j$ , of  
 $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_k) +$   
 $\text{Cost}(M_{k+1} \times M_{k+2} \times \dots \times M_j) +$   
 $r_i C_k C_j$
- \* Write  $\text{Cost}(i,j)$  to denote  $\text{Cost}(M_i \times M_{i+1} \times \dots \times M_j)$



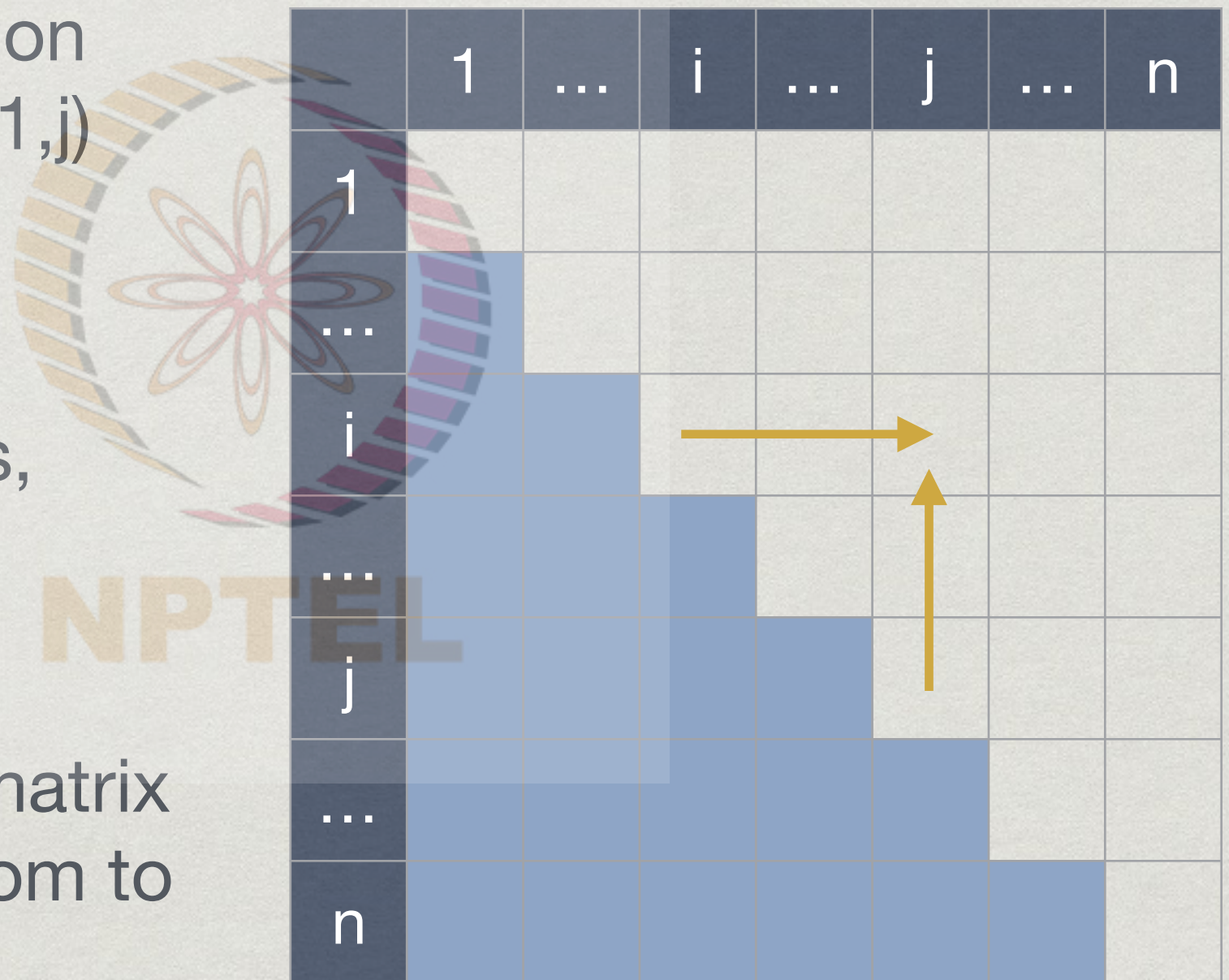
# Final equation

- \*  $\text{Cost}(i,i) = 0$  — No multiplication to be done
- \*  $\text{Cost}(i,j) = \min \text{ over } i \leq k < j$   
[  $\text{Cost}(i,k) + \text{Cost}(k+1,j) + r_i c_k c_j$  ]
- \* Note that we only require  $\text{Cost}(i,j)$  when  $i \leq j$



# Subproblem dependency

- \*  $\text{Cost}(i,j)$  depends on  $\text{Cost}(i,k)$ ,  $\text{Cost}(k+1,j)$  for all  $i \leq k < j$
- \* Can have  $O(n)$  dependent values, unlike LCS, LCW
- \* Start with main diagonal and fill matrix by columns, bottom to top, left to right





# MMCost(M1,...,Mn), DP

```
def MMCost(R,C):  
    # R[0..n-1],C[0..n-1] have row/column sizes  
  
    for r in range(len(R)):  
        MMC[r][r] = 0  
  
    for c in range(1,len(R)):      # c = 1,2,...n-1  
        for r in range(c-1,-1,-1):# r = c,c-1,...,0  
            MMC[r][c] = infinity   # Something large  
            for k in range(r,c)    # k = r,r+1,...,c-1  
                subprob = MMC[r][k] + MMC[k][c] +  
                           R[r]C[k]C[c]  
  
            if subprob < MMC[r][c]:  
                MMC[r][c] = subprob
```



# Complexity

- \* As with LCS, we to fill an  $O(n^2)$  size table
- \* However, filling  $MMCost[i][j]$  could require examining  $O(n)$  intermediate values
- \* Hence, overall complexity is  $O(n^3)$