Dynamic Programming

Course Code: CSC 2211 Course Title: Algorithms

Dept. of Computer Science
Faculty of Science and Technology

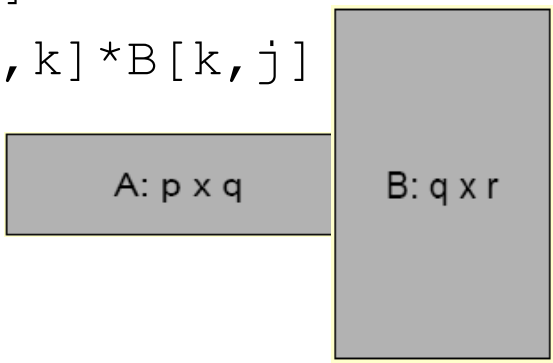
Lecturer No:		**Week No:**	**09**	**Semester:**	**Summer 2019-202**
Lecturer:	*Name & email*				

Lecture Outline

1. Matrix Chain Multiplication (MCM)
2. Longest Common Subsequence (LCS)

Review: Matrix Multiplication

```
Matrix-Multiply(A,B):
   1   if columns[A] != rows[B] then
   2      error "incompatible dimensions"
   3   else{
   5      for i = 1 to rows[A] do
   6         for j = 1 to columns[B] do{
   7            C[i,j] = 0
   8            for k = 1 to columns[A] do
   9               C[i,j] = C[i,j]+A[i,k]*B[k,j]
   10        }
   11  }
   12  return C
```



- Time complexity = $O(pqr)$,
  - where $|A|=p\times q$ and $|B|=q\times r$

Multiplying Matrices

- Two matrices, ***A*** **–** ***n×m*** matrix and ***B*** **–** ***m×k*** matrix, can be multiplied to get ***C*** with dimensions ***n×k,*** using ***n×m×k*** scalar multiplications

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} = \begin{pmatrix} \dots & \dots & \dots \\ \dots & c_{22} & \dots \\ \dots & \dots & \dots \end{pmatrix} \qquad c_{i,j} = \sum_{l=1}^{m} a_{i,l} \cdot b_{l,j}$$

- **<u>Problem</u>**: **Compute** a product of many matrices **efficiently**

Matrix Chain Multiplication [MCM]: The Problem

- **Input**: Matrices $T_1, T_2, \ldots, T_n$, each T_i of size $d_{i-1} \times d_i$
- **Output**: Fully ***parenthesized*** product $T_1 T_2 \ldots T_n$ that minimizes the number of scalar multiplications.
- A product of matrices is fully parenthesized if it is either

  a) a single matrix, or

  b) the product of 2 fully parenthesized matrix products surrounded by parentheses.
- Example: $T_1\ T_2\ T_3\ T_4$ can be fully parenthesized as:

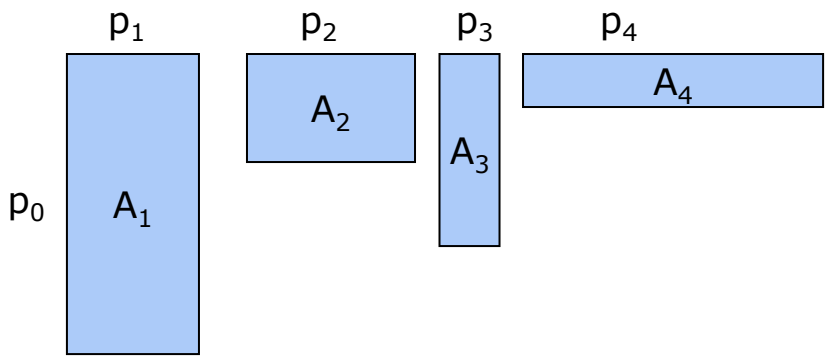
  1. $(T_1\ (T_2\ (T_3\ T_4\ )))$
  2. $(T_1\ ((T_2\ T_3\ )T_4\ ))$
  3. $((T_1\ T_2\ )(T_3\ T_4\ ))$
  4. $((T_1\ (T_2\ T_3\ ))T_4\ )$
  5. $(((T_1\ T_2\ )T_3\ )T_4\ )$

Matrix Chain Multiplication [MCM]: The Problem

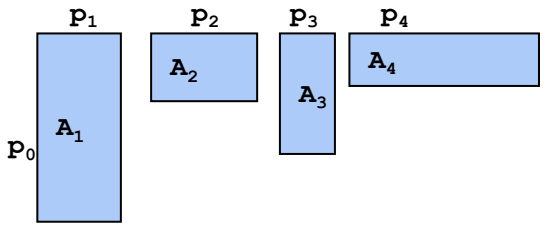
Suppose size of A_i is p_{i-1} by p_i.

$(A_1 \; (\; A_2 \; (\; A_3 \;\; A_4 \;) \;))$: $p_2 \; p_3 \; p_4 + p_1 p_2 p_4 + p_0 p_1 p_4$

$(((A_1 \;\; A_2) \;\; A_3) \;\; A_4 \;)$: $p_0 \; p_1 \; p_2 + p_0 p_2 p_3 + p_0 p_3 p_4$

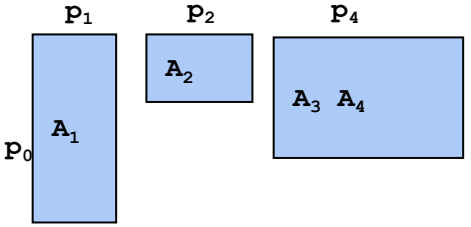


Matrix Chain Multiplication [MCM]: (A1(A2(A3 A4)))



$(A_3\ A_4)$

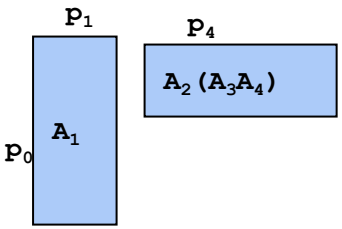
$p_2p_3p_4$ multiplications



$(A_2\ (A_3\ A_4))$

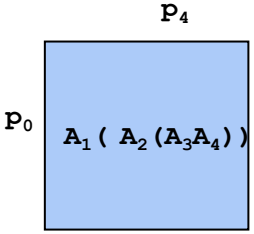
$p_1p_2p_4$ multiplications

Total: $p_2p_3p_4 + p_1p_2p_4 + p_0p_1p_4$

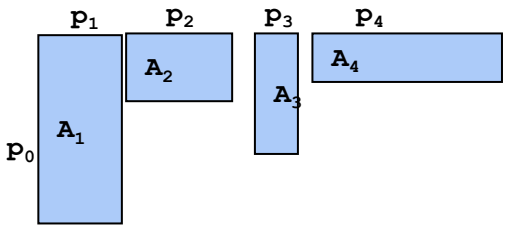


$(A_1\ (A_2\ (A_3\ A_4)))$

$p_0p_1p_4$ multiplications

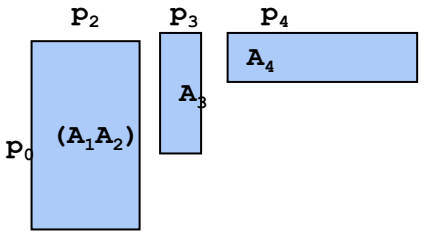


Matrix Chain Multiplication [MCM]: (((A1 A2)A3)A4)



$(A_1\ A_2)$

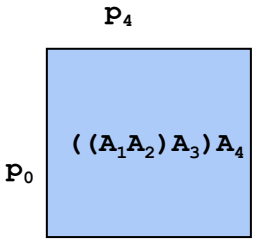
$p_0p_1p_2$ multiplications

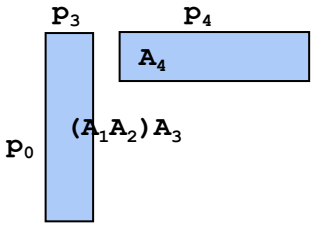


$((A_1\ A_2)A_3)$

$p_0p_2p_3$ multiplications

Total: $p_0p_1p_2 + p_0p_2p_3 + p_0p_3p_4$



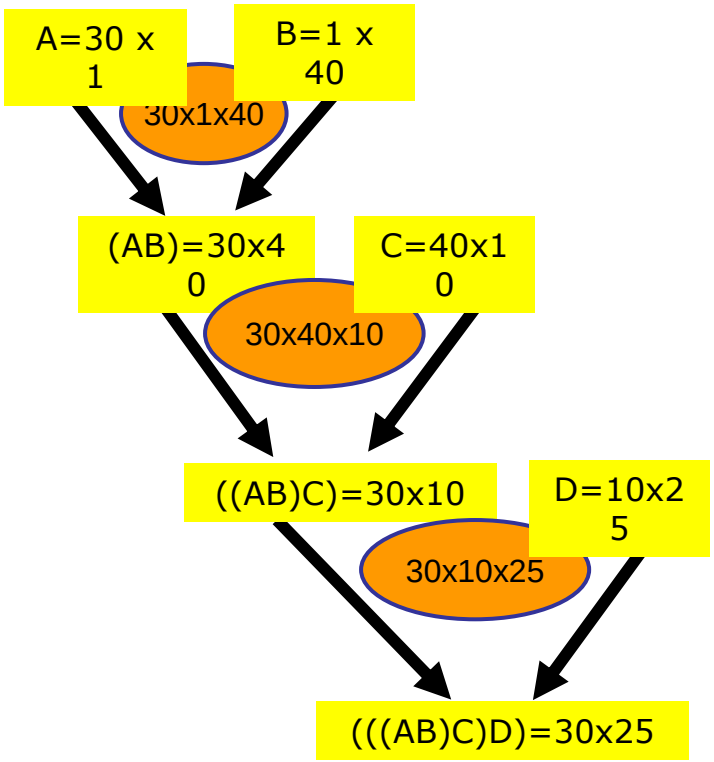


$(((A_1\ A_2)A_3)A_4)$

$p_0p_3p_4$ multiplications

MCM: Parenthesization

- Some Preliminaries
- Matrix multiplication is ***associative***
  - (***AB***)***C*** = ***A***(***BC***)
- The **parenthesization** matters
- Consider ***A***×***B***×***C***×***D***, where
  - ***A*** is **30**×**1**, ***B*** is **1**×**40**,
  - ***C*** is **40**×**10**, ***D*** is **10**×**25**
- Costs:
  - (((***AB***)***C***)***D***) **=120 0 + 1200 0 + 7500 = 20700**

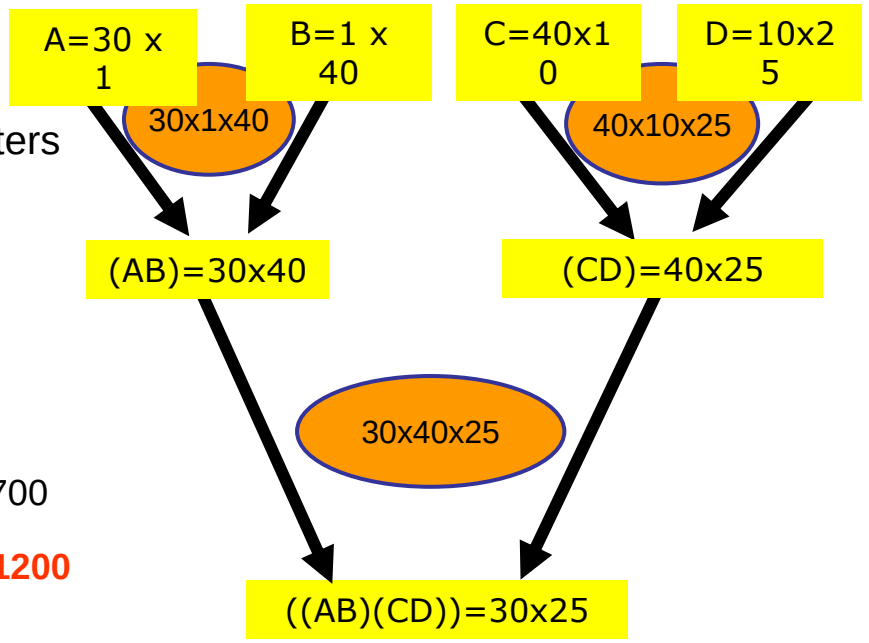


MCM: Parenthesization

- Some Preliminaries
- Matrix multiplication is ***associative***
  - (***AB***)***C*** = ***A***(***BC***)
- The **parenthesization** matters
- Consider ***A***×***B***×***C***×***D***, where
  - ***A*** is **30**×**1**, ***B*** is **1**×**40**,
  - ***C*** is **40**×**10**, ***D*** is **10**×**25**
- Costs:
  - (((***AB***)***C***)***D***)

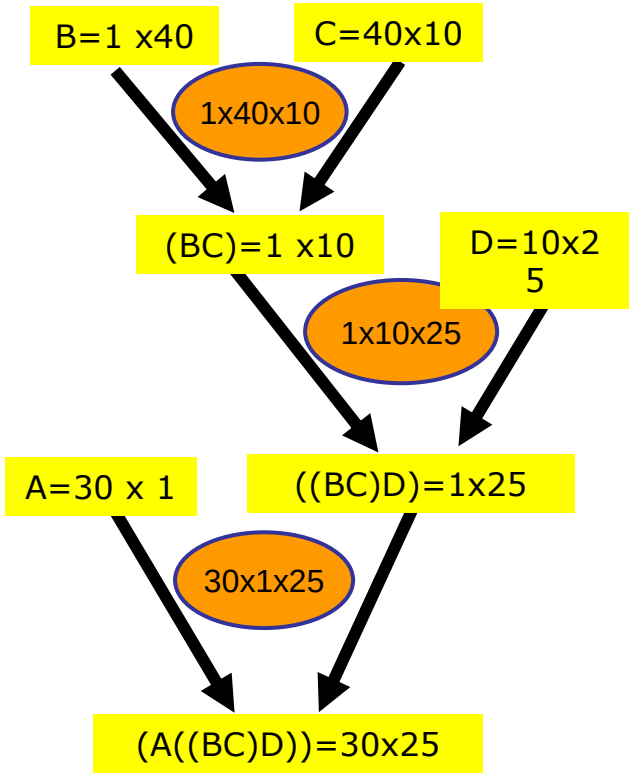
  = 1200 + 12000 + 7500 = 20700
  - ((***AB***)(***CD***))

  =1200 + 10000 + 30000 = 41200



MCM: Parenthesization

- Some Preliminaries
- Matrix multiplication is ***associative***
  - (***AB***)***C*** = ***A***(***BC***)
- The **parenthesization** matters
- Consider ***A***×***B***×***C***×***D***, where
  - ***A*** is **30**×**1**, ***B*** is **1**×**40**,
  - ***C*** is **40**×**10**, ***D*** is **10**×**25**
- Costs:
  - (((***AB***)***C***)***D***)
  = 1200 + 12000 + 7500 = 20700
  - ((***AB***)(***CD***))
  = 1200 + 10000 + 30000 = 41200
  - (***A***((***BC***)***D***))
  =400 + 250 + 750 = 1400



MCM: Parenthesization

- Some Preliminaries
- Matrix multiplication is ***associative***
  - $(AB)C = A(BC)$
- The **parenthesization** matters
- Consider $A \times B \times C \times D$, where
  - A is 30×1, B is 1×40,
  - C is 40×10, D is 10×25
- Costs:
  - $(((AB)C)D)$

  $= 1200 + 12000 + 7500 = 20700$
  - $((AB)(CD))$

  $= 1200 + 10000 + 30000 = 41200$
  - $(A((BC)D))$

  $= 400 + 250 + 750 = 1400$

- We need to optimally parenthesize

$\mathbf{T_1} \times \mathbf{T_2} \times \ldots \mathbf{T_n}$, where $\mathbf{T_i}$ is a $\mathbf{d_{i-1}} \times \mathbf{d_i}$ matrix.

- According to the given example
  - $\mathbf{d} = \{30, 1, 40, 10, 25\}$
  - $\mathbf{T_1} \times \mathbf{T_2} \times \mathbf{T_3} \times \mathbf{T_4}$ where
    - $\mathbf{T_1} = \mathbf{d_{1-1}} \times \mathbf{d_1} = \mathbf{d_0} \times \mathbf{d_1} = 30 \times 1$
    - $\mathbf{T_2} = \mathbf{d_{2-1}} \times \mathbf{d_2} = \mathbf{d_1} \times \mathbf{d_2} = 1 \times 40$
    - $\mathbf{T_3} = \mathbf{d_{3-1}} \times \mathbf{d_3} = \mathbf{d_2} \times \mathbf{d_3} = 40 \times 10$
    - $\mathbf{T_4} = \mathbf{d_{4-1}} \times \mathbf{d_4} = \mathbf{d_3} \times \mathbf{d_4} = 10 \times 25$
- **Costs:**
  - $((\mathbf{T_1}\ \mathbf{T_2})\ \mathbf{T_3})\ \mathbf{T_4} = 20700$
  - $(\mathbf{T_1}\ \mathbf{T_2})\ (\mathbf{T_3}\ \mathbf{T_4}) = 41200$
  - $\mathbf{T_1}\ ((\mathbf{T_2}\ \mathbf{T_3})\ \mathbf{T_4}) = 1400$

MCM: Parenthesization

- Let the number of different parenthesizations, `P(n)`.
- Then, $P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$
- Using ***Generating Function*** we have,
  - `P(n) = C(n-1)`, the $(n-1)^{th}$ ***Catalan Number*** where

$$C(n) = 1/(n+1)C_n^{2n} = \Omega\left(4^n / n^{3/2}\right)$$

- Exhaustively checking all possible parenthesizations take exponential time!

MCM :: Step 1: Characterize Optimal Sub-structure

- Let ***M(i, j)*** be the *minimum* number of multiplications necessary to compute $\mathbf{T_{i..j} = T_i \times \ldots \times T_j}$
- Key observations
  - The outermost parenthesis partitions the chain of matrices ***(i, j)*** at some ***k***, (***i ≤ k < j***): $(\mathbf{T_i \ldots T_k})(\mathbf{T_{k+1} \ldots T_j})$
  - The optimal parenthesization of matrices ***(i, j)*** is also optimal on either side of ***k***; i.e., for matrices (***i, k***) and (***k+1, j***)
  - Within the optimal parenthesization of $\mathbf{T_{i..j}}$,

    (a) the parenthesization of $\mathbf{T_{i..k}}$ must be optimal

    (b) the parenthesization of $\mathbf{T_{k+1..j}}$ must be optimal

    - Why?

MCM :: Step 2: Recursive (Recurrence) Formulation

- Need to find $T_{1..n}$
- Let $M(i, j)$ = minimum # of scalar multiplications needed to compute $T_{i..j}$
- Since $T_{i..j}$ can be obtained by breaking it into $T_{i..k}$ & $T_{k+1..j}$, we have

$$M(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{ M(i,k) + M(k+1,j) + d_{i-1} d_k d_j \} & \text{if } i < j \end{cases}$$

- Note: The sizes of $T_{i..k}$ is $d_{i-1} \times d_k$ and $T_{k+1..j}$ is $d_k \times d_j$,

and $T_{i..k}$ $T_{k+1..j}$ is $d_{i-1} \times d_j$ after d_{i-1} d_k d_j scalar multiplications.

- Let $s(i, j)$ be the value k where the optimal split occurs
- A direct recursive implementation is exponential – a lot of duplicated work.
- But there are only few different subproblems (i, j): one solution for each choice of i and j ($i < j$).

MCM::Step 2: Recursive (Recurrence) Formulation

```
Recursive-Matrix-Chain(d, i, j)
1   if i = j then
2     return 0;
3   M[i, j] = ∞;
4   for k = i to j-1 do
5     q = Recursive-Matrix-Chain(d, i, k) +
          Recursive-Matrix-Chain(d, k+1, j) +
          d_{i-1} d_k d_j
6     if q < M[i, j] then
7               M[i, j] = q ;
                S[i, j] = k ;
8 return M[i, j] ;
```

MCM :: Step 2: Recursive (Recurrence) Formulation

- **Overlapping Subproblems**
- Let T(n) be the time complexity of
  `Recursive-Matrix-Chain(d, 1, n)`
- For n > 1, we have

$$T(n) = 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1)$$

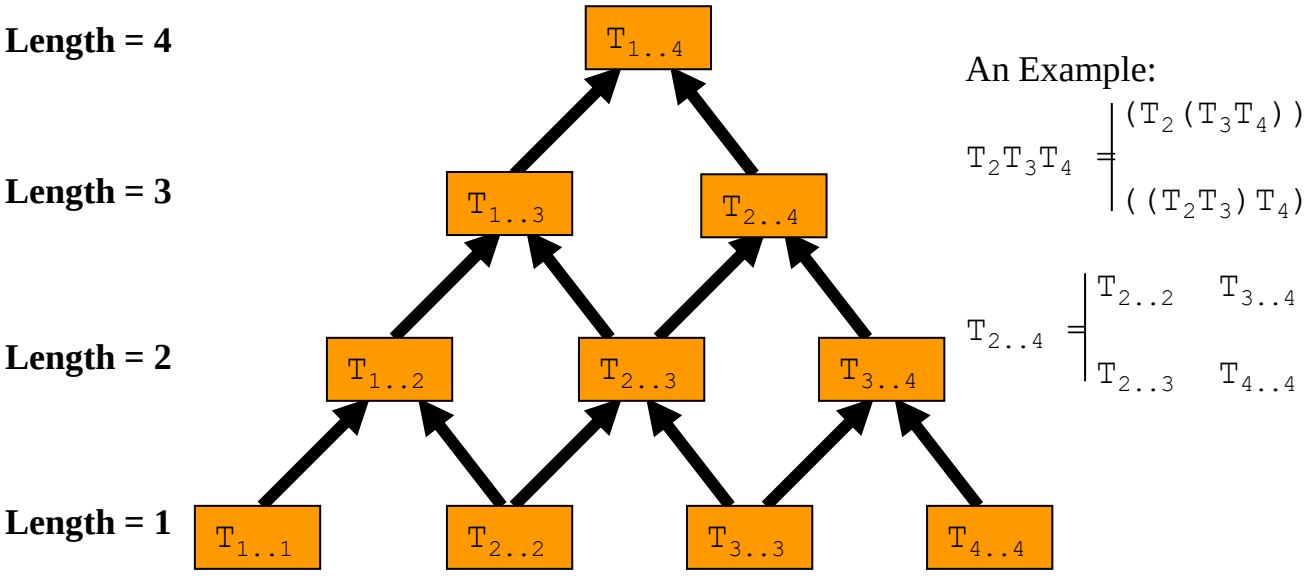
  a) **1** is used to cover the cost of lines 1-3, and 8

  b) 1 is used to cover the cost of lines 6-7

- Using substitution, we can show that $T(n) \geq 2^{n-1}$
- Hence $T(n) = \Omega(2^n)$

MCM :: Step 3: Computing Optimal Costs

- To compute $T_{i..j}$ we need only values for subproblems of length<`i-j`.
- Solve subproblems in the ***increasing length*** of subproblems: first subproblems of length 2, then of length 3 and so on.



MCM :: Step 3: Computing the Optimal Costs

- **Idea:** store the optimal cost $M(i, j)$ for each subproblem in a 2d array $M[1..n, 1..n]$
  - Trivially $M(i, i) = 0$, $1 \le i \le n$
  - To compute $M(i, j)$, where $i - j = L$, we need only values of M for subproblems of length $< L$.
  - Thus we have to solve subproblems in the ***increasing length*** of subproblems: first subproblems of length 2, then of length 3 and so on.
- To reconstruct an optimal parenthesization for each pair (i, j) we record in $s[i, j] = k$ the optimal split into two subproblems (i, k) and $(k+1, j)$

MCM :: Step 3: Computing the Optimal Costs

```
Matrix-Chain-Order( d )

01 n = length[d]-1                    //d is the array of matrix sizes
02 for i = 1 to n do
03     M[i,i] = 0                     // no multiplication for 1 matrix
04 for len = 2 to n do                   // len is length of sub-chain
05     for i = 1 to n-len+1 do                 // i: start of sub-chain
06         j =i+len-1                          // j: end of sub-chain
07     M[i,j] = ∞
08     for k = i to j-1 do
09         q = M[i,k]+M[k+1,j]+d_{i-1} d_k d_j
10             if q < M[i,j] then
11             M[i,j] = q
12             s[i,j] = k

13 return M and s

Time complexity = O(n3 )
```

MCM :: Step 3: Computing the Optimal Costs

- After the execution: ***M [1,n]*** contains the value of an optimal solution and *s* contains optimal subdivisions (choices of ***k***) of any subproblem into two sub-subproblems

- Let us run the algorithm on the six matrices:

Matrix	Dimension
T1	30 X 35
T2	35 X 15
T3	15 X 5
T4	5 X 10
T5	10 X 20
T6	20 X 25

d={ 30, 35, 15, 5, 10, 20, 25 }

- See CLRS Figure 15.3

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03    M[i,i] = 0
04 for len = 2 to n do
05    for i = 1 to n-len+1 do
06       j =i+len-1
07              M[i,j] = ∞
08              for k = i to j-1 do
09                q = M[i,k]+M[k+1,j]+d_{i-1}d_k d_j
10                if q < M[i,j] then
11                       M[i,j] = q
12                       s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

S	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03    M[i,i] = 0
04 for len = 2 to n do
05    for i = 1 to n-len+1 do
06       j =i+len-1
07              M[i,j] = ∞
08              for k = i to j-1 do
09                q = M[i,k]+M[k+1,j]+d_(i-1) d_k d_j
10                if q < M[i,j] then
11                      M[i,j] = q
12                      s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

S	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03     M[i,i] = 0
04 for len = 2 to n do
05     for i = 1 to n-len+1 do
06         j =i+len-1
07                 M[i,j] = ∞
08                 for k = i to j-1 do
09                   q = M[i,k]+M[k+1,j]+d_{i-1} d_k d_j
10                   if q < M[i,j] then
11                           M[i,j] = q
12                           s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1	0	15750				
2		0	2625			
3			0	750		
4				0	1000	
5					0	5000
6						0

len=2, [i,j]=[1,2] to [5,6], k = 1 to 5

M[1,2] = M[1,1]+M[2,2]+d_0*d_1*d_2 = 0+0+30*35*15 = 15750;

M[2,3] = M[2,2]+M[3,3]+d_1*d_2*d_3 = 0+0+35*15* 5 = 2625;

M[3,4] = M[3,3]+M[4,4]+d_2*d_3*d_4 = 0+0+15* 5*10 = 750;

M[4,5] = M[4,4]+M[5,5]+d_3*d_4*d_5 = 0+0+ 5*10*20 = 1000;

M[5,6] = M[5,5]+M[6,6]+d_4*d_5*d_6 = 0+0+10*20*25 = 5000;

S	1	2	3	4	5	6
1		1				
2			2			
3				3		
4					4	
5						5
6						

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03    M[i,i] = 0
04 for len = 2 to n do
05    for i = 1 to n-len+1 do
06       j =i+len-1
07       M[i,j] = ∞
08       for k = i to j-1 do
09         q = M[i,k]+M[k+1,j]+d_{i-1}d_k d_j
10         if q < M[i,j] then
11              M[i,j] = q
12              s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1	0	15750	7875			
2		0	2625	4375		
3			0	750	2500	
4				0	1000	3500
5					0	5000
6						0

len=3, [i,j]=[1,3] to [4,6], k = [1,2] to [4,5]

M[1,3] = M[1,1]+M[2,3]+$d_0*d_1*d_3$ = 0+ 2625 +30*35*5 =7875;

= M[1,2]+M[3,3]+$d_0*d_2*d_3$ = 15750 + 0+30*15*5 =18000;

M[2,4] = M[2,2]+M[3,4]+$d_1*d_2*d_4$ = 0+750+35*15*10 = 6000;

= M[2,3]+M[4,4]+$d_1*d_3*d_4$ = 2625+0+35*5*10 = 4375;

M[3,5] = M[3,3]+M[4,5]+$d_2*d_3*d_5$ = 0+1000+15* 5*20 = 2500;

= M[3,4]+M[5,5]+$d_2*d_4*d_5$ = 750+0+15*10*20 = 3750;

M[4,6] = M[4,4]+M[5,6]+$d_3*d_4*d_6$ = 0+5000+ 5*10*25 = 6250;

= M[4,5]+M[6,6]+$d_3*d_5*d_6$ = 1000+0+ 5*20*25 = 3500;

S	1	2	3	4	5	6
1		1	1			
2			2	3		
3				3	3	
4					4	5
5						5
6						

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03    M[i,i] = 0
04 for len = 2 to n do
05    for i = 1 to n-len+1 do
06       j =i+len-1
07       M[i,j] = ∞
08       for k = i to j-1 do
09          q = M[i,k]+M[k+1,j]+d_{i-1} d_k d_j
10          if q < M[i,j] then
11               M[i,j] = q
12               s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1	0	15750	7875	9375		
2		0	2625	4375	7125	
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

S	1	2	3	4	5	6
1		1	1	3		
2			2	3	3	
3				3	3	3
4					4	5
5						5
6						

len=4, [i,j]=[1,4] to [3,6], k = [1,2,3] to [3,4,5]

M[1,4] = M[1,1]+M[2,4]+d_0*d_1*d_4 = 0+ 4375+30*35*10 =14875;

= M[1,2]+M[3,4]+d_0*d_2*d_4 = 15750+750+30*15*10 =21000;

= M[1,3]+M[4,4]+d_0*d_3*d_4 = 7875+ 0+30*5*10 =9375;

M[2,5] = M[2,2]+M[3,5]+d_1*d_2*d_5 = 0+2500+35*15*20 = 13000;

= M[2,3]+M[4,5]+d_1*d_3*d_5 = 2625+1000+35*5*20 = 7125;

= M[2,4]+M[4,5]+d_1*d_4*d_5 = 4375+1000+35*10*20 = 12375;

M[3,6] = M[3,3]+M[4,6]+d_2*d_3*d_6 = 0+3500+15* 5*25 = 5375;

= M[3,4]+M[5,6]+d_2*d_4*d_6 = 750+5000+15* 10*25 = 9500;

= M[3,5]+M[6,6]+d_2*d_5*d_6 = 2500+0+15* 20*25 = 10000;

Simulation-MCM

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03     M[i,i] = 0
04 for len = 2 to n do
05     for i = 1 to n-len+1 do
06         j =i+len-1
07         M[i,j] = ∞
08         for k = i to j-1 do
09           q = M[i,k]+M[k+1,j]+d_{i-1} d_k d_j
10           if q < M[i,j] then
11                M[i,j] = q
12                s[i,j] = k
13 return M and s
```

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

M	1	2	3	4	5	6
1	0	15750	7875	9375	11875	
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

len=5, [i,j]=[1,5] to [2,6], k = [1,2,3,4] to [2,3,4,5]

M[1,5] = M[1,1]+M[2,5]+d_0*d_1*d_5 = 0+ 7125+30*35*20 = 28125;

= M[1,2]+M[3,5]+d_0*d_2*d_5 = 15750+ 2500+30*15*20 = 27250;

= M[1,3]+M[4,5]+d_0*d_3*d_5 = 7875+ 1000+30*5*20 = 11875;

= M[1,4]+M[5,5]+d_0*d_4*d_5 = 9375+ 0+30*10*20 = 15375;

M[2,6] = M[2,2]+M[3,6]+d_1*d_2*d_6 = 0+5375+35*15*25 = 18500;

= M[2,3]+M[4,6]+d_1*d_3*d_6 = 2625+3500+35*5*25 = 10500;

= M[2,4]+M[5,6]+d_1*d_4*d_6 = 4375+5000+35*10*25 = 18125;

= M[2,5]+M[6,6]+d_1*d_5*d_6 = 0+5375+35*20*25 = 22875;

S	1	2	3	4	5	6
1		1	1	3	3	
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

Simulation-MCM

	0	1	2	3	4	5	6
d	30	35	15	5	10	20	25

```
Matrix-Chain-Order( d )

01 n = length[d]-1
02 for i = 1 to n do
03    M[i,i] = 0
04 for len = 2 to n do
05    for i = 1 to n-len+1 do
06       j =i+len-1
07       M[i,j] = ∞
08       for k = i to j-1 do
09         q = M[i,k]+M[k+1,j]+d_{i-1}d_k d_j
10         if q < M[i,j] then
11              M[i,j] = q
12              s[i,j] = k
13 return M and s
```

M	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

len=6, [i,j]=[1,6], k = [1,2,3,4,5]

M[1,6] = M[1,1]+M[2,6]+d_0*d_1*d_6 = 0+ 10500+30*35*25 = 36750;

= M[1,2]+M[3,6]+d_0*d_2*d_6 = 15750+ 5375+30*15*25 = 32375;

= M[1,3]+M[4,6]+d_0*d_3*d_6 = 7875+ 3500+30*5*25 = 15125;

= M[1,4]+M[5,6]+d_0*d_4*d_6 = 9375+ 5000+30*10*25 = 21875;

= M[1,5]+M[6,6]+d_0*d_5*d_6 = 11875+ 0+30*20*25 = 26875;

S	1	2	3	4	5	6
1		1	1	3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

MCM :: Step 4: Constructing Optimal Solution

- To get the optimal solution $T_{1..6}$, `s[]` is used as follows:

$T_{1..6}$

$= (T_{1..3}\ T_{4..6})$; since **s[1,6] = 3**

$= ((T_{1..1}\ T_{2..3})(T_{4..5}\ T_{6..6}))$
; since **s[1,3] =1** and **s[4,6]=5**

$= ((T_1\ (T_2\ T_3))((T_4\ T_5)T_6))$

MCM can be solved in $O(n^3)$ time

Matrix Chain Multiplication Problem

- Running time
  - It is easy to see that it is $O(n^3)$ (three nested loops)
  - It turns out it is also $\Omega(n^3)$
- Thus, a reduction from exponential time to polynomial time.

Memoization

- ***Memoization*** **is one way to deal with overlapping subproblems**
  - After computing the solution to a subproblem, store it in a table
  - Subsequent calls just do a table lookup
- **Can modify recursive algorithm to use memoziation**
- **If we prefer recursion we can structure our algorithm as a recursive algorithm**:

```
MemoMCM(i,j)
1.   if i = j then return 0
2.   else if M[i,j] < ∞ then return M[i,j]
3.   else for k := i to j-1 do
4.           q :=  MemoMCM(i,k)+
                                  MemoMCM(k+1,j)
     + d_{i-1} d_k d_j
5.                  if q < M[i,j] then
6.                     M[i,j] := q
7.        return M[i,j]
```

- **Initialize all elements to ∞ and call `MemoMCM(i,j)`**

Memoization

- Memoization:
  - Solve the problem in a ***top-down*** fashion, but record the solutions to subproblems in a table.
- Pros and cons:
  - ☹ Recursion is usually slower than loops and uses stack space
  - ☺ Easier to understand
  - ☺ If not all subproblems need to be solved, you are sure that only the necessary ones are solved

Longest Common Subsequence The Problem

- Given two sequences
  - $X = \langle x_1, x_2, \ldots, x_m \rangle$
  - $Z = \langle z_1, z_2, \ldots, z_k \rangle$
- **Z** is a ***subsequence*** of **X** if
  - $z_j = x_{i(j)}$ for all $j = 1, \ldots, k$
  - where $\langle i(1), i(2), \ldots, i(k) \rangle$ is ***strictly increasing*** (but not required to be consecutive)
- Examples:
  - Let $X = \langle A, B, C, B, D, A, B \rangle$
  - $\langle A \rangle$, $\langle B \rangle$, $\langle C \rangle$, and $\langle D \rangle$ are subsequences.
  - $\langle C, A \rangle$, $\langle C, B \rangle$, $\langle C, B, A, B \rangle$ are subsequences.
  - How many possible subsequences for a ***n***-element sequence?

Longest Common Subsequence Problem

- **Z** is a ***common*** subsequence of sequences **X** and **Y** if **Z** is a subsequence of both **X** and **Y**.

- Example:
  - Let **X = < A, B, C, B, D, A, B >** and **Y = < B, D, C, A, B, A >**
  - **< A >, <B>, <C>,** and **<D>** are common subsequences.
  - **< C, A >** is, but **< A, C >** is not.
  - **< B, C, A>** is a common subsequence
  - **< B, C, B, A >** is the longest common subsequence.

- Example:
  - **x = "sariempiolcewe"**
  - **y = "westigmupsalrte"**

Longest Common Subsequence Problem

- **LCS**:
- Input: two sequences $x[1..m]$ and $y[1..n]$
- Output: longest common subsequence of x and y (denoted $LCS(x, y)$)
- Brute-force algorithm:
  - For every subsequence of x, check if it is a subsequence of y
  - 2^m subsequences of x to check against n elements of y : $O(n\ 2^m)$

LCS: Optimal Sub-structure

- The i^{th} ***prefix*** of $X = \langle x_1, x_2, \ldots, x_m \rangle$ is denoted $X_i = \langle x_1, x_2, \ldots, x_i \rangle$
  - X_0 is the ***empty*** sequence
  - X_m is the ***whole*** sequence X

- Theorem 15.1 (Optimal Sub-structure of LCS)

- Let $X=\langle x_1, x_2, \ldots, x_m \rangle$, $Y=\langle y_1, y_2, \ldots, y_n \rangle$, and $Z=\langle z_1, z_2, \ldots, z_k \rangle$ be $LCS(X, Y)$.

(1) If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is $LCS(X_{m-1}, Y_{n-1})$

(2) if $x_m \neq y_n$, then $z_k \neq x_m$ implies Z is $LCS(X_{m-1}, Y)$

(3) if $x_m \neq y_n$, then $z_k \neq y_n$ implies Z is $LCS(X, Y_{n-1})$

LCS: Optimal Substructure

- We make Z to be empty and proceed from the ends of $X_m = "x_1\ x_2\ \ldots x_m"$ and $Y_n = "y_1\ y_2\ \ldots y_n"$
  - If $x_m = y_n$, append this symbol to the beginning of Z, and find optimally $\mathbf{LCS}(X_{m-1},\ Y_{n-1})$
  - If $x_m \neq y_n$,
    - Skip either a letter from X
    - or a letter from Y
    - Decide which decision to do by comparing

      $\mathbf{LCS}(X_m,\ Y_{n-1})$ and $\mathbf{LCS}(X_{m-1},\ Y_n)$
  - Starting from beginning is equivalent.

LCS: Optimal Substructure

- **LCS** has an optimal sub-structure defined by ***prefixes*** of **X** and **Y**
- The sub-problems of finding **LCS** (X_{m-1}, Y_n) and **LCS** (X_m, Y_{n-1}) share a common sub-sub-problem of finding **LCS** (X_{m-1}, Y_{n-1}). They are overlapping.
- **Simplify:** just worry about **LCS** length for now
  - Define **c[i, j]** = length of **LCS** (X_i, Y_j)
  - So **c[m, n]** = length of **LCS (X, Y)**

LCS: Recurrence

- Define **`c[i, j]`** = length of **LCS** of **`x[1..i], y[1..j]`**

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

- Note that the conditions in the problem restrict sub-problems (if $\mathbf{x_i} = \mathbf{y_i}$ we consider $\mathbf{x_{i-1}}$ and $\mathbf{y_{i-1}}$, etc)

- Use **`b[i, j]`** to remember where to extract an element of an **LCS**.

LCS: Recurrence

```
int lcsRec(int i, int j) {
  if (i==0 || j==0) return 0;
  else if (x[i] == y[j])
    return lcsRec(i-1,j-1) + 1;
  else
    return max(lcsRec(i-1,j),lcsRec(i,j-1));
}
```

Recurrence for time complexity:

$T(2n)=T(2n-2)+1$ *if* $x[n]=y[n]$

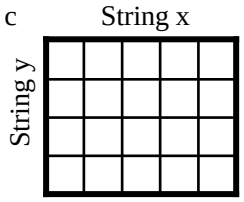
$T(2n)=2T(2n-1)$ *if* $x[n]!=y[n]$

$T(2n)=1$ *if* $n=0$

$$
\begin{aligned}
T(2n) &= 2T(2n-1) \\
&= 2(2T(2n-2)) \\
&= 4T(2n-2) \\
&= 4(2T(2n-3)) \\
&= 8T(2n-3) \\
&= 2^{i}T(2n-i) \\
&= 2^{2n} = 4^{n}
\end{aligned}
$$

LCS: Recurrence

```
int lcsMemo(int i, int j) {
  if (c[i][j] != -1) return c[i][j]
  else if (x[i] == y[j]) {
    c[i][j] = lcsMemo(i-1,j-1) + 1
    return c[i][j]
  }
  else {
    c[i][j]=max(lcsMemo(i-1,j),lcsMemo(i,j-1))
    return c[i][j]
  }
}
```



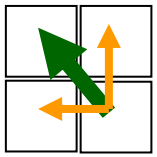
$T(n) = O(n^2)$

LCS: Computing Length

```
LCS-Length(X, Y, m, n)
1   for i←1 to m do
2       c[i,0] ← 0
3   for j←0 to n do
4       c[0,j] ← 0
5   for i←1 to m do
6       for j←1 to n do
7           if x_i = y_j then
8               c[i,j] ← c[i-1,j-1]+1
9               b[i,j] ← "copy"
10          else if c[i-1,j] ≥ c[i,j-1]
    then
11                  c[i,j] ← c[i-1,j]
12                  b[i,j] ← "skipX"
13                  else
14                  c[i,j] ← c[i,j-1]
15                  b[i,j] ← "skipY"
16  return c, b
```

LCS: Example

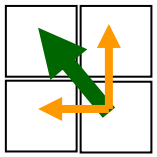
$$c[i,j] = \begin{cases} 0, & \text{if } i=0,\ j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$



		X	a	b	b	c	a	a	c
	Y	0	0	0	0	0	0	0	0
a		0							
c		0							
c		0							
b		0							
c		0							
c		0							
a		0							

LCS: Example

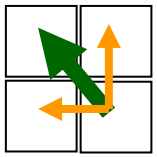
$$c[i,j] = \begin{cases} 0, & \text{if } i=0,\ j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$



		a	b	b	c	a	a	c
	0	0	0	0	0	0	0	0
a	0							
c	0							
c	0							
b	0							
c	0							
c	0							
a	0							

LCS: Example

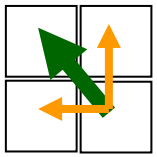
$$c[i,j] = \begin{cases} 0, & \text{if } i=0,\ j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$



		a	b	b	c	a	a	c
	0	0	0	0	0	0	0	0
a	0							
c	0							
c	0							
b	0							
c	0							
c	0							
a	0							

LCS: Example

$$c[i,j] = \begin{cases} 0, & \text{if } i=0,\ j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1],c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$



		a	b	b	c	a	a	c
	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1
c	0	1	1	1	2	2	2	2
c	0	1	1	1	2	2	2	3
b	0	1	2	2	2	2	2	3
c	0	1	2	2	3	3	3	3
c	0	1	2	2	3	3	3	4
a	0	1	2	2	3	4	4	4

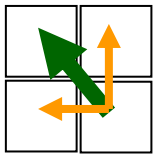
Constructing an LCS

```
Print-LCS(b, X, i, j)

1  if i = 0 or j = 0 then

2        return

3  if b[i,j] = "copy" then

4        Print-LCS(b,X,i-1,j-1)

5              print x[i]

6 elseif b[i,j]="skipX" then

7        Print-LCS(b,X,i-1,j)

8  else

         Print-LCS(b,X,i,j-1)
```

length[X] = m, length[Y] = n,
Call Print-LCS(b, X, n, m) to construct LCS
Time complexity: O(m+n).

LCS: Example

$$c[i,j] = \begin{cases} 0, & \text{if } i=0,\ j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1],c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_i \end{cases}$$



		a	b	b	c	a	a	c
	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1
c	0	1	1	1	2	2	2	2
c	0	1	1	1	2	2	2	3
b	0	1	2	2	2	2	2	3
c	0	1	2	2	3	3	3	3
c	0	1	2	2	3	3	3	4
a	0	1	2	2	3	4	4	4

Longest Common Subsequence

- There is a need to quantify how similar they are:
  - Comparing DNA sequences in studies of evolution of different species
  - Spell checkers
  - Editing

Books

1. ***Introduction to Algorithms, Third Edition, Thomas H. Cormen, Charle E. Leiserson, Ronald L. Rivest, Clifford Stein (CLRS).***
2. *Fundamental of Computer Algorithms, Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran (HSR)*

References

- https://www.radford.edu/~nokie/classes/360/dp-matrix-parens.html
- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Dynamic/chainMatrixMult.htm
- https://www.ics.uci.edu/~eppstein/161/960229.html
- CLRS: 15.2, 15.4