

(MATROID)

## STRATEGY GREEDY

Solution  $\leftarrow \emptyset$

for  $i \leftarrow 1$  to  $n$  do

**SELECT** the next input  $x$ .

If  $\{x\} \cup \text{Solution}$  is

**FEASIBLE** then

Solution  $\leftarrow \text{COMBINE}(\text{Solution}, x)$

---

Select appropriately finds the next input to be considered.

A feasible solution satisfies the constraints required for the output.

Combine enlarges the current solution to include a new input.

## STRATEGY DIVIDE - AND - CONQUER

**DIVIDE** problem  $P$  into smaller problems  $P_1, P_2, \dots, P_k$ .

**SOLVE** problems  $P_1, \dots, P_k$  to obtain solutions  $s_1, s_2, \dots, s_k$

**Combine** solution  $s_1, s_2, \dots, s_k$  to get the final solution.

---

Subproblems  $P_1, P_2, \dots, P_k$  are solved **recursively** using divide-and-conquer.

---

## 2 STRATEGY DIVIDE-AND-CONQUER

- Divide Problem  $P$  into smaller problem  $P_1, P_2, \dots, P_k$ .
- Solve problems  $P_1, P_2, \dots, P_k$  to obtain solutions  $S_1, S_2, \dots, S_k$
- Combine solution  $S_1, S_2, \dots, S_k$  to get the final solution.

Subproblems  $P_1, P_2, \dots, P_k$  are solved recursively using divide-and-conquer.

‘

Examples: Quicksort and mergesort.

---

### 3 STRATEGY GREEDY

Solution  $\leftarrow \Phi$

for  $i \leftarrow 1$  to  $n$  do

**SELECT** the next input  $x$ .

If  $\{x\} \cup \text{Solution}$  is **FEASIBLE** then

solution  $\leftarrow \text{COMBINE}(\text{Solution}, x)$

- **SELECT** appropriately finds the next input to be considered.
- A **FEASIBLE** solution satisfies the constraints required for the output.
- **COMBINE** enlarges the current solution to include a new input.

Examples: Max finding, Selection Sort, and Kruskal's Smallest Edge First algorithm for Minimum Spanning Tree.

---

## 4 STRATEGY DYNAMIC PROGRAMMING

- Fibonacci Numbers:

$$F_n = F_{n-1} + F_{n-2}$$

$$F_1 = F_0 = 1.$$

- Recursive solution requires exponential time:  
has **overlapping subproblems**.
- **Bottom-up** iterative solution is linear –  
**compute once, store, and use many times.**

---

## 4.1 Matrix Sequence Multiplication

- eg. 1:

$$A_{30 \times 1} \times B_{1 \times 40} \times C_{40 \times 10} \times D_{10 \times 25} \times E_{25 \times 1}$$

- Left to right evaluation requires more than 12K multiplications.
- $(A \times ((B \times C) \times (D \times E)))$  needs only 690 multiplications (minimum needed).
- **Greedy Algorithm: Largest Common Dimension First**

- eg. 2:

$$A_{1 \times 2} \times B_{2 \times 3} \times C_{3 \times 4} \times D_{4 \times 5} \times E_{5 \times 6}$$

- Largest Common Dimension First imposes following order:

$$(A \times (B \times (C \times (D \times E))))$$

which needs 240 multiplications.

— Best order:

$$(((A \times B) \times C) \times D) \times E)$$

which needs 68 multiplications.

— **Another Greedy Algorithm: Smallest Common Dimension First** but did not work for eg. 1.

---

## 4.2 Divide and Conquer Solution

**Input:**  $A_1 * A_2 \dots * A_n$   
 $d_0 * d_1 \quad d_1 * d_2 \dots \quad d_{n-1} * d_n$

**Output:** A parenthesization of the input sequence resulting in minimum number of multiplications needed to multiply the  $n$  matrices.

- **Subgoal:** Ignore Structure of Output (order of parenthesization), focus on obtaining a numerical solution (minimum number of multiplications)
- Define  $M[i,j] =$  the minimum number of multiplications needed to compute

$$A_i * A_{i+1} * \dots * A_j$$

for  $i \leq j \leq n$

- Subgoal is to obtain  $M[1, n]$ .

e.g. For,

$$A1_{30x1} \ X \ A2_{1x40} \ X \ A3_{40x10} \ X \ A4_{10x25} \ X \ A5_{25x1}$$

$$M[1,1] = 0, M[1,2] = 1200, M[1,5] = 690$$

---

### 4.3 Recursive Formulation of $M[i, j]$

- $A1 \times A2 = (A1) \times (A2)$ 
  - Partition at  $k=1$ : Subproblems  $(A1)$  and  $(A2)$
  - cost of  $(A1)$  is  $M[1, 1]$  and that of  $(A2)$  is  $M[2, 2]$
  - cost of combining  $(A1)$  and  $(A2)$  into one is  $d_0 * d_1 * d_2$ .
  - $M[1, 2] = M[1, 1] + M[2, 2] + d_0 * d_1 * d_2$ .
  - $M[1, 2] = 0 + 0 + 1200 = 1200$ .
- $A2 \times A3 \times A4 = (A2) \times (A3 \times A4) \ (k=2)$   
Or,  $= (A2 \times A3) \times A4 \ (k=3)$ .
  - $k = 2$ : cost =  $M[2, 2] + M[3, 4] + d_1 * d_2 * d_4$
  - $k = 3$ : cost =  $M[2, 3] + M[4, 4] + d_1 * d_3 * d_4$
  - $M[2, 4] = \min(M[2, 2] + M[3, 4] + d_1 * d_2 * d_4, M[2, 3] + M[4, 4] + d_1 * d_3 * d_4)$

$$\text{In short, } M[2, 4] = \min_{2 \leq k \leq 3} (M[2, k] + M[k, 4] + d_1 d_k d_4)$$

- A2 X A3 X A4 X A5  
 $= (\text{A2}) \times (\text{A3} \times \text{A4} \times \text{A5}) \ (k=2)$   
 Or,  $= (\text{A2} \times \text{A3}) \times (\text{A4} \times \text{A5}) \ (k=3)$   
 Or,  $= (\text{A2} \times \text{A3} \times \text{A4}) \times (\text{A5}) \ (k=4)$

$$M[2, 5] = (M[2, 2] + M[3, 5] + d_1 d_2 d_5, M[2, 3] + M[4, 5] + d_1 d_3 d_5, M[2, 4] + M[5, 5] + d_1 d_4 d_5)$$

- In general, by factoring  $(A_i * A_{i+1} * \dots * A_j)$  at  $k$ th index position into  $(A_i * A_{i+1} * \dots * A_k)$  and  $(A_{k+1} * \dots * A_j)$  need  $M[i, k] + M[k+1, j]$  multiplications and creates matrices of dimensions  $d_{i-1} * d_k$  and  $d_k * d_j$ . These two matrices need additional  $d_{i-1} * d_k * d_j$  multiplications to combine.

---

4.4

## Recursive Formula and Time Taken



- Recursively,

$$M[i, j] =$$

$$\min_{i \leq k \leq j-1} (M[i, k] + M[k+1, j] + d_{i-1}d_kd_j)$$

$$M(i, i) = 0$$

- **Optimal Substructure**



- 
- We can recursively solve for

$$M[1, n] =$$

$$\min_{1 \leq k \leq n-1} (M[1, k] + M[k+1, n] + d_0 d_k d_n) \\ = \min[M[1, 1] + M[2, n] + d_0 d_1 d_n,$$

$$M[1, 2] + M[3, n] + d_0 d_2 d_n,$$

$$M[1, 3] + M[4, n] + d_0 d_3 d_n,$$

:

$$M[1, n-1] + M[n, n] + d_0 d_{n-1} d_n$$

- Time Complexity:

$$T_n = n + T_1 + T_{n-1} \\ + T_2 + T_{n-2} \\ + T_3 + T_{n-3} \\ \vdots \\ + T_{n-2} + T_2 \\ + T_{n-1} + T_1$$

$$T_n = n + 2T_1 + 2T_2 + \cdots + 2T_{n-1} \quad (1)$$

$$T_{n-1} = n - 1 + 2T_1 + 2T_2 + \cdots + 2T_{n-2} \quad (2)$$

Subtracting (I)-(II) yields

$$T_n - T_{n-1} = 1 + 2T_{n-1}$$

$$T_n = 1 + 3T_{n-1}$$

$$= 1 + 3(1 + 3T_{n-2})$$

$$T_n = 1 + 3 + 3^2 + 3^3 + \cdots + 3^{n-1}T_1$$

$$= 1 + 3 + 3^2 + 3^3 + \cdots + 3^{n-2}$$

- 
- Recursive Solution is exponential time  $\Omega(3^{n-2})$
  - Space  $O(n)$  stack depth.
  - **Overlapping subproblems:** e.g. Recursion tree for  $M[1, 4]$ .  
26 recursive calls for just 10 subproblems  
 $M[1, 1], M[2, 2], M[3, 3], M[4, 4], M[1, 2], M[2, 3], M[3, 4]$

---

So we turn to dynamic Programming,

- the same formulation
- approach the problem bottom-to-top
- find a suitable table to store the sub-solutions.

How many sub-solutions do we have?

$$M[1, 1], M[2, 2], M[3, 3] \dots, M[n, n] \quad n$$

$$M[1, 2], M[2, 3], \dots, M[n-1, n] \quad n-1$$

$$M[1, 3], M[2, 4], \dots, M[n-2, n] \quad n-2$$

⋮

$$M[1, n-1], M[2, n] \quad 2$$

$$M[1, n] \quad 1$$

$$\frac{n(n-1)}{2}$$

⇒ we need  $O(n^2)$  space

⑨

## MATRIX ORDER

M, FACTOR : MATRIX

for  $i \leftarrow 1$  to  $n$  do  $M[i, j] \leftarrow 0$   
 // main diagonal

for diagonal  $\leftarrow 1$  to  $\underline{n}$  do

for  $i \leftarrow 1$  to  $n$ -diagonal do

$$j = i + \text{diagonal}$$

$$M[i, j] = \min_{i \leq k \leq j-1} [M[i, k] + M[k+1, j] + d_{i-1} d_k d_j]$$

factor[i, j] = k that  
 gave the min value for  
 $M[i, j]$ .

and for

and for

$$(A_1 \times A_2 \times A_3 \times A_4 \times A_5) : 10 \times 25 \quad 25 \times 1$$

(1)

$A_1 : 30 \times 1$

$A_2 : 1 \times 40$

$A_3 : 40 \times 10$

1	2	3	4	5	
1	0	1200 (1)	700 (1)	1400 (1)	690 (1)
2	0	400 (2)	650 (3)	660 (3)	
3	0		100 00 (3)	650 (3)	
4		0		250 (4)	
5			0		

$$M[1,2] = \min_{1 \leq k \leq 1} [M[i,k] + M[k+1,j] + d_{i-1} d_k d_j]$$

$$= \min [M[1,1] + M[2,2] + d_0 d_1 d_2]$$

$$= 0 + 0 + 30 \times 1 \times 40$$

$$= 1200$$

$$M[1,3] = \min_{1 \leq k \leq 3-1} [M[i,k] + M[k+1,j] + d_{i-1} d_k d_j]$$

$$= \min [M[1,1] + M[2,3] + d_0 d_1 d_3,$$

$$M[1,2] + M[3,3] + d_0 d_2 d_3]$$

$$= \min [0 + 400 + \frac{30 \times 1 \times 10}{1200 + 0 + 30 \times 40 \times 10}]$$

$$= \min [700, 12000 + 1200] = 700$$

---



## 4.5 Matrix Parenthesization Order

M,Factor: Matrix

```
for  $i \leftarrow 1$  to  $n$  do  $M[i, i] \leftarrow 0$   
/* main diagonal */
```

```
for diagonal  $\leftarrow 1$  to  $n - 1$  do  
for  $i \leftarrow 1$  to  $n - \text{diagonal}$  do
```

$j = i + \text{diagonal}$

$$M[i, j] = \min_{i \leq k \leq j-1} (M[i, k] + M[k+1, j] + d_{i-1} d_k d_j)$$

Factor[i, j] = k that gave the minimum value  
for  $M[i, j]$ .

endfor

endfor

---

#### 4.6 Work out

$$\begin{aligned} & A1_{30x1} \times A2_{1x40} \times A3_{40x10} \times A4_{10x25} \times A5_{25x1} \\ M[1,2] &= \min_{1 \leq k \leq 1} [M[i, k] + M[k + 1, j] + d_{i-1}d_kd_j] \\ &= \min[M[1, 1] + M[2, 2] + d_0d_1d_2] \\ &= 0 + 0 + 30 * 1 * 40 \\ &= 1200 \end{aligned}$$

$$\begin{aligned} M[1,3] &= \min_{1 \leq k \leq 3-1} [M[i, k] + M[k + 1, j] + d_{i-1}d_kd_j] \\ &= \min[M[1, 1] + M[2, 3] + d_0d_1d_3, \\ &\quad M[1, 2] + M[3, 3] + d_0d_2d_3] \\ &= \min[0 + 400 + 30 * 1 * 10, \\ &\quad 1200 + 0 + 30 * 40 * 10] \\ &= \min[700, 12000 + 1200] \\ &= 700 \end{aligned}$$

$$\begin{aligned}
M[2,4] &= \min[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j] \\
2 \leq k &\leq 3 \\
&= \min[M[2,2] + M[3,4] + d_1d_2d_4, \\
&\quad M[2,3] + M[4,4] + d_1d_3d_4] \\
&= \min[0 + 10000 + 1 * 40 * 25, 400 + 0 + 1 * 10 * 25] \\
&= \min[10100, 650] = 650
\end{aligned}$$

$$\begin{aligned}
M[3,5] &= \min[M[i,k] + M[k+1,j] + d_{i-1}d_kd_j] \\
3 \leq k &\leq 4 \\
&= \min[M[3,3] + M[4,5] + d_2d_3d_5, \\
&\quad M[3,4] + M[5,5] + d_2d_4d_5] \\
&= \min[0 + 250 + 40 * 10 * 1, 10000 + 0 + 40 * 25 * 1] \\
&= \min[650, -] = 650
\end{aligned}$$

$$\begin{aligned}
M[1,4] &= \min[M[1,1] + M[2,4] + d_0d_1d_4, \\
&\quad M[1,2] + M[3,4] + d_0d_2d_4, \\
&\quad M[1,3] + M[4,4] + d_0d_3d_4] \\
&= \min[0 + 650 + 30 * 1 * 25, \\
&\quad 1200 + 10000 + 30 * 40 * 25, \\
&\quad 700 + 0 + 30 * 10 * 25] \\
&= \min[1400, -, -] = 1400
\end{aligned}$$

---

## 5 DYNAMIC PROGRAMMING REQUIREMENTS

**Requirements:**  a) Optimal Substructure  
 b) Overlapping subproblem

**Steps:**  1)  Characterize the structure of an optimal solution  
 2) Formulate a recursive solution  
 3) Compute the value of *an* opt. solution bottom-up. (get value rather than the structure)  
4) Construct an optimal solution (structure) from computed information.

  **Memoization:** Top-down, compute and store first time, reuse subsequent times.