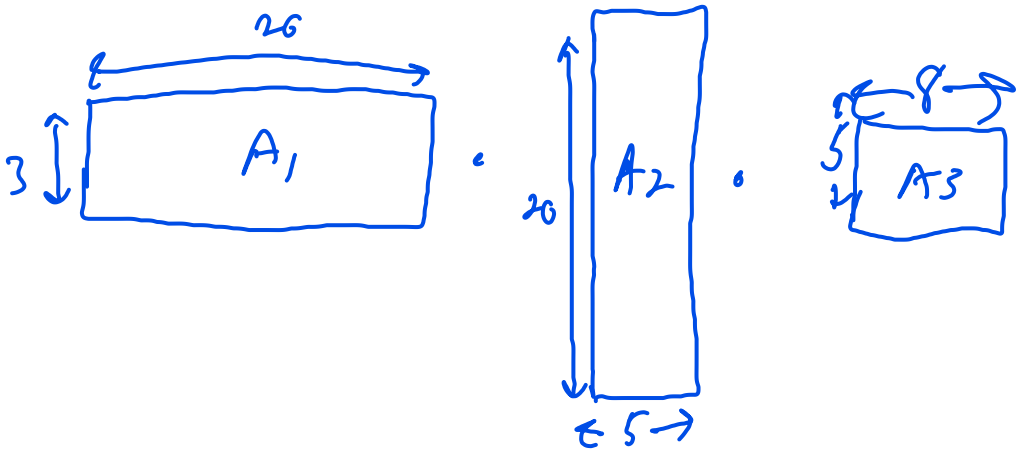
Suppose we are given a sequence/chain of n matrices $A_1, A_2, \ldots, A_n$, and we are interested in computing the product $A_1 \cdot A_2 \cdots A_n$.

Note that if this product can be computed, then there are $n+1$ values $p_0, p_1, \ldots, p_n$ such that matrix A_i is a $p_{i-1} \times p_i$ matrix.

- Example: $n = 3, p_0 = 3, p_1 = 20, p_2 = 5, p_3 = 8$.



Computing $A_1 \cdot A_2$ takes $3 \cdot 20 \cdot 5$ multiplications and results in a 3×5 matrix.

In general, computing $A_i \cdot A_{i+1}$ takes $p_{i-1} \cdot p_i \cdot p_{i+1}$ multiplications and results in a $p_{i-1} \times p_{i+1}$ matrix.

Matrix multiplication is associative, and therefore the order in which we multiply the matrices does not affect the resulting matrix.

- Computing $(A_1 \cdot A_2) \cdot A_3$ results in the same matrix as computing $A_1 \cdot (A_2 \cdot A_3)$.

That being said, the order in which we do the multiplications can greatly affect the total number of scalar multiplications performed by the algorithm.

- Computing $(A_1 \cdot A_2) \cdot A_3$ takes $3 \cdot 20 \cdot 5 + 3 \cdot 5 \cdot 8 = 300 + 120 = 420$ multiplications.
- Computing $A_1 \cdot (A_2 \cdot A_3)$ takes $20 \cdot 5 \cdot 8 + 3 \cdot 20 \cdot 8 = 800 + 480 = 1280$ multiplications.

Given that the order in which we multiply the matrices can have a huge impact on the running time of the algorithm, it often is worth the time in practice to compute an optimal "parenthesization" of the matrices that minimizes the number of scalar multiplications needed to compute the product.

What is the running time of the brute force algorithm which checks the cost of each possible parenthesization and returns the best one?

- The algorithm is correct, but we can show that the number of parenthesizations is $\Omega(2^n)$. Too slow.

What about a "greedy" strategy which iteratively tries to find a "good" pair of matrices to multiply?

- We can construct counterexamples which show that a greedy strategies can fail.

$A_1: 1\times 20$, $A_2: 20\times 21$, $A_3: 21\times 5$

Eliminate largest # first: $A_1 \cdot (A_2 \cdot A_3)$

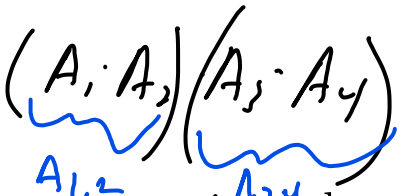
20×5

$20 \cdot 21 \cdot 5 + 1 \cdot 20 \cdot 5 = 2200$

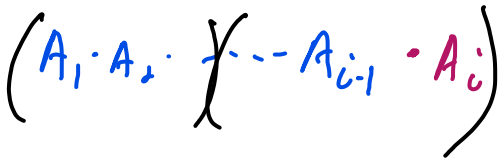
Other way: $(A_1 \cdot A_2) \cdot A_3$

1×21

$1 \cdot 20 \cdot 21 + 1 \cdot 21 \cdot 5 = 525$



How about a dynamic programming algorithm? Can we express an optimal solution as a combination of optimal solutions for some set of subproblems?



Let $A_{i,j} = A_i \cdot A_{i+1} \cdots A_j$ for $i \leq j$ (note we are interested in computing $A_{1,n}$).

Suppose an optimal parenthesization for $A_{i,j}$ splits this matrix at k, so

$$A_{i,j} = (A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$$

$p_{i-1} \times p_k$ $\quad$ $p_k \times p_j$

Notice that this parenthesization will compute $A_{i,k}$ and $A_{k+1,j}$ and then multiply these matrices together using $p_{i-1} \cdot p_k \cdot p_j$ scalar multiplications.

When computing $A_{i,k} = A_i \cdots A_k$, an optimal solution for $A_{i,j}$ must use an optimal parenthesization of $A_i \cdots A_k$ (otherwise we contradict the assumption that the solution was optimal).

Let $m[i, j]$ denote the minimum number of scalar multiplications to compute $A_{i,j}$. Therefore the cost of splitting $A_{i,j}$ at k is

$A_1 \cdot A_2 \cdot A_3 A_4$

$$\underbrace{m[i, k]}_{\text{to compute } A_{i,k}} + \underbrace{m[k + 1, j]}_{\text{to compute } A_{k+1,j}} + \underbrace{p_{i-1} \cdot p_k \cdot p_j}_{\text{Multiply 2 Solutions together.}}$$

We have the following recurrence relation:

$m[i, i] = 0 \quad \forall i.$

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

$$m[i, j] = \min_{i \le k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} \cdot p_k \cdot p_j \}$$

We can see that the running time of the straight forward recursive algorithm is $\Omega(2^n)$, but there are only $\Theta(n^2)$ subproblems (recursive algorithm can call the same subproblem multiple times).

Bottom-up dynamic programming algorithm:

$A_1(A_2A_3)$

$n=4,\ p_0=2,\ p_1=100,\ p_2=4,\ p_3=10,\ p_4=3$

$A_1: 2\times100,\ A_2: 100\times4,\ A_3: 4\times10,\ A_4: 10\times3$

i \ j	1	2	3	4
1	0	800	880	940
2	X	0	4000	1320
3	X	X	0	120
4	X	X	X	0

$m[1,3] \Rightarrow k=1,2$

$k=1: m[1,1] + m[2,3] + 2\cdot100\cdot10$
$= 0 + 4000 + 2000 = 6000$

$k=2: m[1,2] + m[3,3] + 2\cdot4\cdot10$
$= 800 + 0 + 80 = 880$

$m[2,4] \Rightarrow k=2,3$

$k=2: m[2,2] + m[3,4] + 100\cdot4\cdot3$
$0 + 120 + 1200 = 1320$

$k=3: m[2,3] + m[4,4] + 100\cdot10\cdot3$
$4000 + 0 + 3000 = 7000$

$m[1,4] \Rightarrow k=1,2,3$

$k=1: m[1,1] + m[2,4] + 2\cdot100\cdot3$
$0 + 1320 + 600 = 1920$

$k=2: m[1,2] + m[3,4] + 2\cdot4\cdot3$
$800 + 120 + 24 = 944$

$k=3: m[1,3] + m[4,4] + 2\cdot10\cdot3$
$880 + 0 + 60$
$= 940$