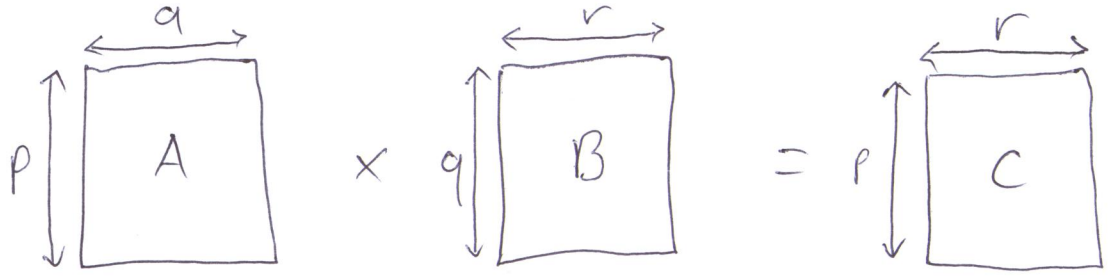
Matrix Chain Multiplication

A: $p \times q$ matrix
B: $q \times r$ matrix
} $C = AB$: $p \times r$ matrix



C has pr entries. Each entry can be computed in $O(q)$ time, so C can be computed in $O(pqr)$ time.

Consider 3 matrices

$A_1: 10 \times 100$ $A_2: 100 \times 5$ $A_3 = 5 \times 50$

How to compute $A_1 A_2 A_3$?

① compute $A_1 A_2$: cost $= 10 \times 100 \times 5 = 5{,}000$
compute $\underbrace{(A_1 A_2)}_{10 \times 5} \underbrace{A_3}_{5 \times 50}$: cost $= 10 \times 5 \times 50 = 2{,}500$ +
7,500 total

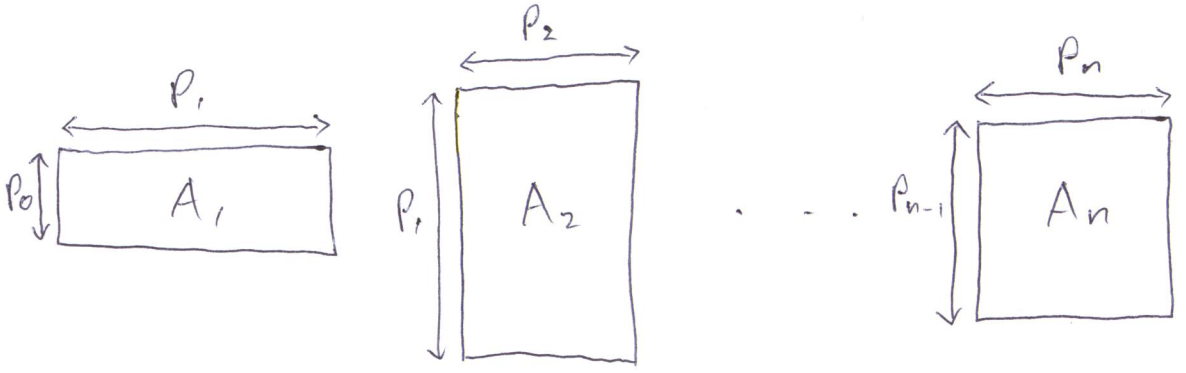
② compute $A_2 A_3$: cost $= 100 \times 5 \times 50 = 25{,}000$
compute $\underbrace{A_1}_{10 \times 100} \underbrace{(A_2 A_3)}_{100 \times 50}$: cost $= 10 \times 100 \times 50 = 50{,}000$ +
75,000 total

⇒ The order can make a huge difference.

Given matrices $A_1, A_2, \ldots, A_n$

We only care about the sizes: $p_0, p_1, \ldots, p_n$

Matrix A_i has p_{i-1} rows and p_i columns.



Input: positive integers $p_0, p_1, \ldots, p_n$

Output: Minimum cost for multiplying $A_1, \ldots, A_n$.

① Structure of the optimal solution:

Consider the best order to compute $AA_2 \ldots A_n$

In the last step, we multiply

$$\underbrace{(A_1 \ldots A_k)}_{\text{already computed}}\underbrace{(A_{k+1} \ldots A_n)}_{\text{already computed}} \quad \text{for some } 1 \le k \le n-1$$

How did we compute $A_1 \ldots A_k$ and $A_{k+1} \ldots A_n$?

In the best order.

$$\begin{aligned} \text{cost} = {} & \text{min cost to compute } A_1 \ldots A_k \\ & + \text{min cost to compute } A_{k+1} \ldots A_n \\ & + p_0 \cdot p_k \cdot p_n \end{aligned}$$

② Recurrence

Let $m(i,j)$ = minimum cost to compute $A_i \dots A_j$.

We want to compute $m(1,n)$.

If we know k, then

$$m(i,j) = m(i,k) + m(k+1,j) + p_{i-1} p_k p_j$$

But we don't know k.

$\Rightarrow$ Try all values $i \le k \le j-1$ and pick the best!

$$m(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k \le j-1} \{ m(i,k) + m(k+1,j) + p_{i-1} p_k p_j & \text{if } i \neq j \end{cases}$$

③ Compute bottom-up

Compute in order of length $(j-i+1)$: shortest first.

Algorithm MatrixChainMultiply(P):

```
for i ← 1 to n do m[i,i] ← 0
for l ← 2 to n do
  for i ← 1 to n-l+1 do
    j ← i+l-1
    m[i,j] ← ∞
    for k ← i to j-1 do
      m[i,j] ← min(m[i,j], m[i,k] + m[k+1,j] + P[i-1] × P[k] × P[j])
return m[1,n]
```

- Correct? Yes, by earlier arguments.
- Terminates? Yes
- Efficient? 3 nested for-loops $\Rightarrow O(n^3)$

More carefully:

$$\sum_{l=2}^{n}\sum_{i=1}^{n-l+1}\sum_{k=i}^{i+l-2} 1 = \sum_{l=2}^{n}\sum_{i=1}^{n-l+1}(i+l-2-i+1)$$

$$= \sum_{l=2}^{n}\sum_{i=1}^{n-l+1}(l-1)$$

$$= \sum_{l=2}^{n}(n-l+1)(l-1)$$

$$= \sum_{x=1}^{n-1}(n-x)x \qquad (x = l-1)$$

$$= \sum_{x=1}^{n-1} nx - \sum_{x=1}^{n-1} x^2$$

$$= n\cdot\sum_{x=1}^{n-1} x - \sum_{x=1}^{n-1} x^2$$

$$= n\cdot\tfrac{1}{2}(n-1)n - \tfrac{1}{6}(n-1)n(2n-1)$$

$$= \tfrac{1}{2}n(n-1)\cdot\left(n - \tfrac{1}{3}(2n-1)\right)$$

$$= \tfrac{1}{2}n(n-1)\cdot\tfrac{1}{3}(3n-2n+1)$$

$$= \tfrac{1}{6}n(n-1)(n+1)$$

$$= \tfrac{1}{6}n(n^2-n+n-1)$$

$$= \tfrac{1}{6}n(n^2-1)$$

$$= \tfrac{1}{6}(n^3-n)$$

$$= \Theta(n^3)$$