CLASS P:

$$P = \bigcup_k TIME(n^k)$$

P — all problems which can be solved in polynomial time.

For instance, Sorting $\in$ TIME(nlogn)

linear search $\in$ TIME(n).

Q: If you face a hard problem which seems like polynomially solvable, but naive brute-force won't work, what would be a viable solution?

Dynamic Programming!

It is different from divide and conquer. the difference is that you store intermediate solutions in an explicit* way (which means you might need to use more space).

We will go over two examples in detail:

① Matrix chain multiplication

② A_{CFG}

Matrix Chain Multiplication

Given $M_1, M_2, M_3, \cdots, M_n$, compute the product $M_1 \cdot M_2 \cdot \cdots \cdot M_n$, where M_i has dimension $d_{i-1} \times d_i$ (or, M_i has d_{i-1} rows and d_i columns).

<u>Ex</u>. $A_{n \times n} \cdot B_{n \times n} \cdot X_{n \times 1}$

$(A \cdot B) \cdot X \Rightarrow n^3 + n^2$ scalar multiplications

$A \cdot (B \cdot X) \Longrightarrow n^2 + n^2$ scalar multiplications

<u>problem</u>: Parenthesize the product $M_1 M_2 \cdots M_n$ in a way that minimizes the number of scalar multiplications.

- We could take a look at the scanned handout, to see the differences.

- Exhaustive search is not efficient to be practical!

$P(n)$ — # of alternative parenthesization of n matices

$$(M_1 \cdots M_k) \downarrow (M_{k+1} \cdots M_n)$$

$$P(n) = \begin{cases} 1, & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k), & \text{if } n \geq 2. \end{cases}$$

$P(n)$ — nth Catalan number

- $P(n) = \frac{1}{n}\binom{2n-2}{n-1} \geq \frac{4^{n-1}}{2n^2 - n} = \Omega\left(\frac{4^n}{n^2}\right).$

$n = 10$, $P(n) \geq \Omega\left(\frac{4^{10}}{10^2}\right) \sim 10{,}000 = 10^4$

$n = 20$, $P(n) \geq \Omega\left(\frac{4^{20}}{20^2}\right) \sim 10^9$

$n = 40$, no computer can enumerate all solutions.

Dynamic Programming

- Matrix Chain Multiplication

$$(M_1 \cdot M_2 \cdots M_k) \cdot (M_{k+1} \cdots M_n)$$

Divide part

- Let $m[i,j]$ be the number of multiplications performed using an optimal parenthesization of

$$M_i M_{i+1} \cdots M_j \qquad (i<j)$$

$$\left(\underset{d_{i-1}\times d_i}{\underset{\downarrow}{M_i}} M_{i+1} \cdots M_{k-1} \underset{d_{k-1}\times d_k}{\underset{\downarrow}{M_k}}\right) \left(\underset{d_k\times d_{k+1}}{\underset{\downarrow}{M_{k+1}}} \cdots \underset{d_{j-1}\times d_j}{\underset{\downarrow}{M_j}}\right)$$

$$\begin{cases} m[i,j] = \min\limits_{k} \left\{ \overset{m[i,k]}{m_{i,k}} + \overset{m[k+1,j]}{m_{k+1,j}} + d_{i-1} \cdot d_k \cdot d_j \right\} & 1 \le i \le k < j \le n \\ m[i,i] = 0 \end{cases}$$

Suppose we find $m[-,-]$, where is the optimal solution?

$\underline{m[1,n]}$

– Example: M_1 – 20×10
M_2 – 10×50
M_3 – 50×5
M_4 – 5×30

m[-,-]:

i \ j	1	2	3	4
1	0	10,000		
2		0	2,500	
3			0	7,500
4				0

pass 1

pass 0

Let's look at m[1,3]

$$m[1,3] = \min \begin{cases} k=1, & m[1,1] + m[2,3] + 20 \cdot 10 \cdot 5 \\ & = 0 + 2500 + 1000 \\ & = 3,500 \\ k=2, & m[1,2] + m[3,3] + 20 \cdot 50 \cdot 5 \\ & = 10,000 + 0 + 5000 \\ & = 15,000 \end{cases}$$

$\underbrace{M_1}_{20\times10}\underbrace{(M_2M_3)}_{10\times5}$

$\underbrace{(M_1M_2)}_{20\times50}\underbrace{M_3}_{50\times5}$

$= 3,500$

Try to fill out m[2,4] by yourself, using 20 minutes.