SE220 Lecture 3&4(Preview) Notes

I. Asymptotic Notation

i. O-Notation

$O(g(n))$ { $f(n)$: **there exist positive constants** c and n_0 such that $f(n)\epsilon[0, cg(n)]$ **for all** $n \geqslant n_0$ }.

ii. Ω-Notation

$\Omega(g(n))$ ={ $f(n)$: **there exist positive constants** c and n_0 such that $cg(n)\epsilon[0, f(n)]$ **for all** $n \geqslant n_0$ }

iii. Θ-Notation

$\Theta(g(n))$ ={ $f(n)$: **there exists positive constants** c_1 , c_2 and n_0 such that $f(n)\epsilon[c_1 g(n),\ c_2 g(n)]$ **for all** $n \geqslant n_0$ }

II. Divide-and-Conquer Example – **Square Matrices Multiplication**

Choose two 2×2 matrices A and B as example to illustrate how this algorithm is applied to compute $C = AB$ that matrix C is also a square one.

We have known that for an arbitrary entry in $n \times n$ matrix C, it has the formula

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

If we compute the multiplication directly, which needs 3 **for** loop, and the running time is $T(n) = \Theta(n^3)$. However, we have got an idea of partitioning the matrix to compute multiplication in linear algebra, which can be also considered an another way to solve this problem.

If we have $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, and $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$, and $C = AB$ is supposed to be:

$$C=\begin{pmatrix}A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22}\\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22}\end{pmatrix}$$

Let C have the form $C=\begin{pmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{pmatrix}$, which corresponds to the form we just computed:

$$\begin{aligned}C_{11}&=A_{11}B_{11}+A_{12}B_{21}\\ C_{12}&=A_{11}B_{12}+A_{12}B_{22}\\ C_{21}&=A_{21}B_{11}+A_{22}B_{21}\\ C_{22}&=A_{21}B_{12}+A_{22}B_{22}\end{aligned}$$

We can consider each entry in matrix A and B as a submatrix with size $\frac{n}{2}$, and what we need to do is to compute eight $\frac{n}{2}\times\frac{n}{2}$ multiplications and four additions, which can be completed with divide-and-conquer algorithm by recursion(call the function itself). Thus, we can obtain the running time

$$T(n)=\begin{cases}\Theta(1) & n=1\\ 8T\left(\frac{n}{2}\right)+\Theta(1) & n>1\end{cases}=\begin{cases}c_0 & n=1\\ 8T\left(\frac{n}{2}\right)+c_1 & n>1\end{cases}$$

$$\begin{aligned}T(n)&=c_1+\sum_{i=1}^{\log_2 n-1}8^i\cdot\frac{c_1}{2}+8^{\log_2 n}\cdot c_0\\ &=c_1+\frac{1}{2}\sum_{i=1}^{\log_2 n-1}8^i c_1+c_0n^3\\ &=\Theta(n^3)\end{aligned}$$

It illustrates that there is no difference in time complexity in terms of both ways.

III. Most-common-used **Master Theorem**

Proof For a given recurrence equation of the form $T(n)=aT\left(\frac{n}{b}\right)+cn^d$, where $a>0$, $b>1$, c and d are constants. Use recursion tree to solve this equation, we have:

$$\begin{aligned}T(n)&=cn^d+\sum_{i=1}^{\log_b n}cn^d\left(\frac{a}{b^d}\right)^i\\ &=cn^d\left[1+\sum_{i=1}^{\log_b n}\left(\frac{a}{b^d}\right)^i\right]\end{aligned}$$

When $a = b^d \, (d = \log_b a)$, $T(n) = cn^d(1 + \log_b n) = \Theta(n^d \log_b n)$;

When $a \neq b^d \, (d \neq \log_b a)$,

$$
\begin{aligned}
T(n) &= cn^d \left\{ 1 + \frac{\frac{a}{b^d}\left[\left(\frac{a}{b^d}\right)^{\log_b n} - 1\right]}{\frac{a}{b^d} - 1} \right\} \\
&= cn^d \left[1 + \frac{\frac{a}{b^d}\left(n^{\log_b a - d} - 1\right)}{\frac{a}{b^d} - 1} \right]
\end{aligned}
$$

If $a < b^d \, (d > \log_b a)$, $n^{\log_b a - d} - 1 < 0, \frac{a}{b^d} - 1 < 0 \Rightarrow 1 + \frac{a}{b^d} \cdot \frac{n^{\log_b a - d} - 1}{\frac{a}{b^d} - 1} > 0$

Thus, $T(n) = \Theta(n^d)$.

If $a > b^d \, (d < \log_b a)$, $n^{\log_b a} > n^d$

Thus, $T(n) = \Theta\left(n^{\log_b a}\right)$

Finally, this most-common master theorem can be restated as:

$$
T(n) = aT\left(\frac{n}{b}\right) + cn^d = \begin{cases} \Theta(n^d \log_b n), \ a = b^d \\ \Theta(n^d), \ a < b^d \\ \Theta\left(n^{\log_b a}\right), \ a > b^d \end{cases}
$$