

Amortized cost : 一種表示法

17-4a

用“每一個人的平均”去表示“全部加總”

Example:

A copy machine

Single operation

OP₁ 0.5~1 sec.
OP₂ 0.5~1 sec.
⋮
OP₄₉₉ 0.5~1 sec.
OP₅₀₀ 0.5~1 + 120 sec.
OP₅₀₁ 0.5~1 sec.
OP₅₀₂ 0.5~1 sec.
⋮
OP₁₀₀₀ 0.5~1 + 120 sec.

best-case: 0.5

(沒意義, 廣告詞)

worst-case: 121 $\Rightarrow T(n) \leq 121 * n$

(有品質保證, 但太悲觀)

amortized: 1.24 $\Rightarrow T(n) \leq 1.24 * n$

(這個表示法最好)

Note: amortized \neq average-case

\uparrow $0.75 + 120 * (1/500)$

Example: A k-bit binary counter (single operation)

17-4b

best-case : 1

worst-case : k

$\Rightarrow T(n) \leq k * n$

amortized : 2 (最好的表示法) $\Rightarrow T(n) \leq 2 * n$ (tighter!)

Why $T(n)/n$, not $T(n)$?

* Usually, we compare two DSs according to their single operation running time.

(How many times an OP will be performed is unknown.)

* Simple $\left\{ \begin{array}{ll} \text{① } 1.24 \text{ sec. per page} & (\checkmark) \\ \text{② } 620 \text{ sec. for } 500 \text{ pages} & (\times) \\ \text{③ } 1.24n \text{ sec. for } n \text{ pages} & (\times) \end{array} \right.$

Aggregate Method

17-5a

OP_1	t_1
OP_2	t_2
OP_3	t_3
\vdots	\vdots
OP_n	t_n

① Compute $T(n) = \sum t_i$

worst-case total time
(as tight as possible)

② Compute ①/n

Problem: It may be not easy to compute $\sum t_i$ tightly!

Accounting Method

17-5b

Operation	Amortized cost	Actual cost	Δ credit
X	a_x	t_x	
Y	① a_y	t_y	② How
Z	Assign a_z	t_z	(specific object)

③ prove credit ≥ 0 (for any n)

$$\Rightarrow \sum_{\text{付}} a_i \geq \sum_{\text{花}} t_i \quad (\sum_{\text{付}} a_i = \sum_{\text{花}} t_i + C)_{\text{存}}$$

④ $T(n) = O(\sum_{\text{花}} a_i)$

⑤ Compute ④/n

Problem: ① and ② are not easy!

Potential Method

17-6a

$D_0 \xrightarrow{OP_1} D_1 \xrightarrow{OP_2} D_2 \xrightarrow{OP_3} \dots \xrightarrow{OP_n} D_n$

① Define $\Phi(D_i) \sim$ credit after OP_i

② prove $\Phi(D_i) - \Phi(D_0) \geq 0$ for any $i \rightarrow \sum a_i \geq \sum t_i$ (付 \geq 花)

OP	Amortized cost	Actual cost	Δ credit
X	a_x	t_x	
Y	a_y	t_y	$\Phi(D_i) - \Phi(D_{i-1})$
Z	a_z	t_z	

$$a_i = t_i + [\Phi(D_i) - \Phi(D_{i-1})]$$

付 花 存款 變化

④ $T(n) = O(\sum a_i)$

⑤ Compute ④/n

17-6b

a_i	t_i	Δ credit	D_i	$\Phi(D_i)$
			D_0	200
7	4	3	D_1	203
6	2	4	D_2	207
7	13	-6	D_3	201
6	1	5	D_4	206
6	8	-2	D_5	204
	$\sum t_i = ?$			

$$\sum a_i \geq \sum t_i \text{ (付 } \geq \text{ 花)}$$

② prove $\Phi(D_i) - \Phi(D_0) \geq 0$

④ $T(n) \leq \sum a_i \leq 7n$

⑤ $T(n)/n \leq 7n/n \leq 7$

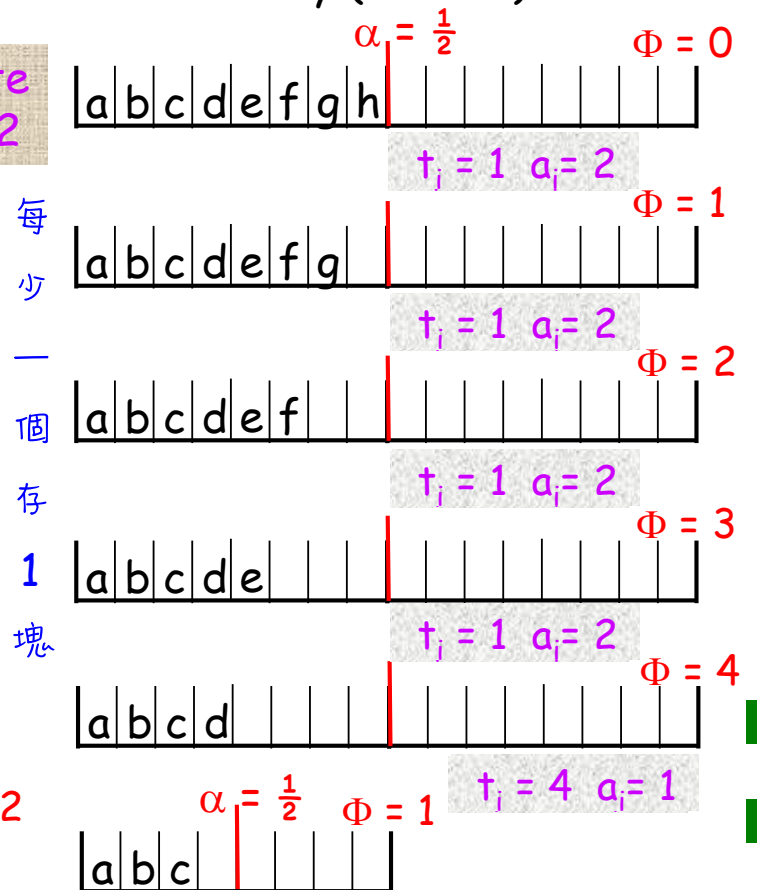
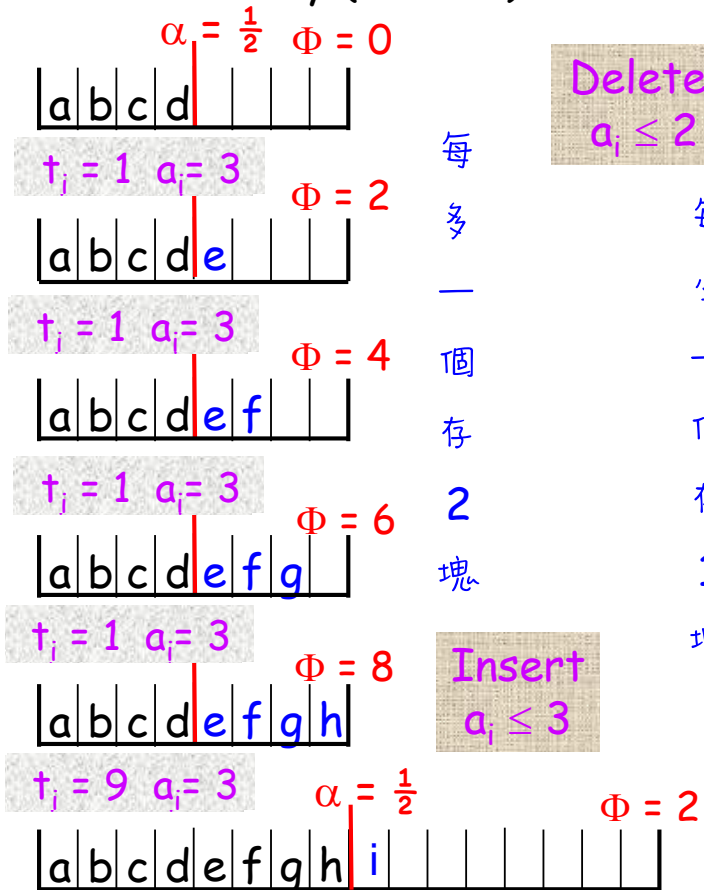
$$a_i = t_i + [\Phi(D_i) - \Phi(D_{i-1})]$$

付 花 存款 變化

Insertion Only ($\alpha \geq 1/2$)

Deletion Only ($\alpha \leq 1/2$)

17-10a

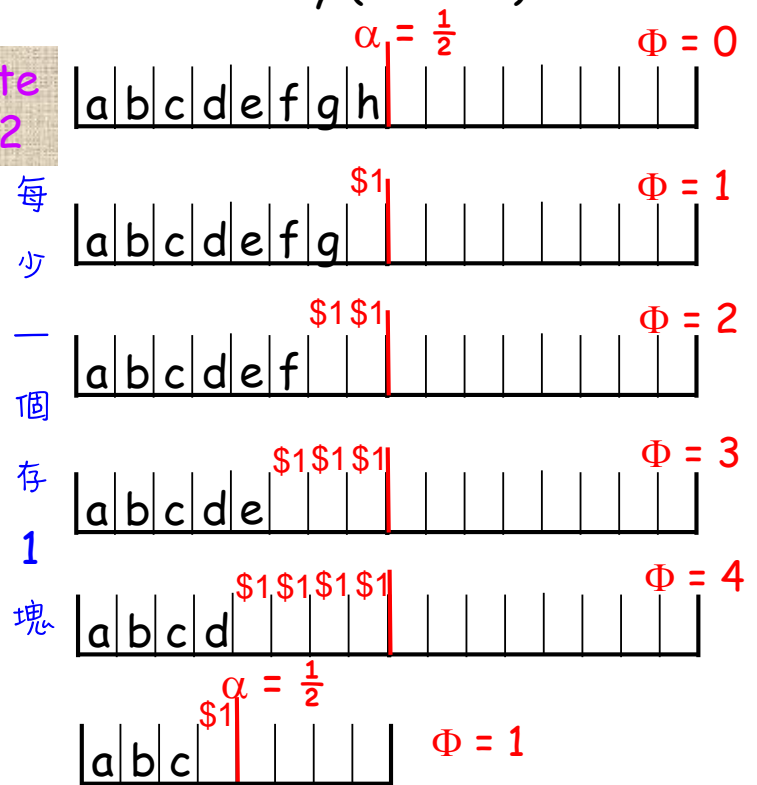
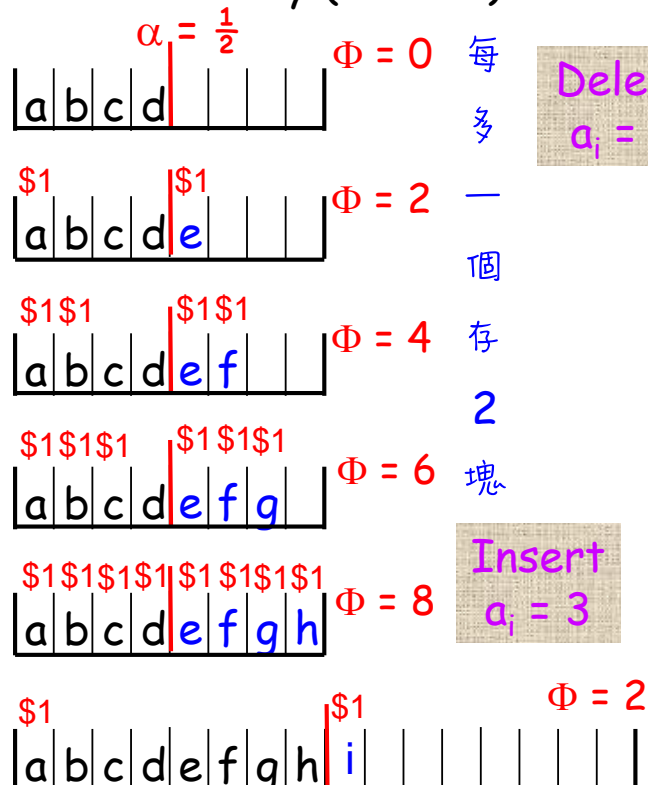


Another viewpoint -- Accounting Method

17-10b

Insertion Only ($\alpha \geq 1/2$)

Deletion Only ($\alpha \leq 1/2$)



Amortized cost: 一種表示法

17-13a

用“每一個人的平均”去表示“全部加總”

Selection of a DS (for a library)

	worst-case	amortized	
DS ₁	$O(n)$	$O(1)$	good for lib (or a group of users)
DS ₂	$O(\lg n)$	$O(\lg n)$	good for a single user

* 如果需要多次呼叫, amortized 比 worst-case 有意義

* single-operation worst-case 好 \Rightarrow 每次都很好 (快)

* single-operation amortized cost 好

\Rightarrow 整體表現好 (偶爾很差 (慢))

\Rightarrow 快快...快, 很慢, 快, 快...快, 很慢, ...
(存存...存, 花, 存, 存, ...存, 花, ...)

Why amortized analysis?

for $i = 1$ to n do
 OP_i

17-13b

Time: $t_1, t_2, t_3, \dots, t_n$

(1) Analysis

(a) Traditional

✓ $t_i = O(f(n))$

✓ $T(n) = n \times O(f(n)) = O(n \times f(n))$

(b) Amortized: compute $T(n)$ directly

✓ aggregate method

✓ accounting method

✓ potential method

} 用規律方式付錢, 再用
全部付款的去估實際花費

(e.g., 一年生活費?)

(use when most t_i are small and $f(n)$ occurs only a few times)

for $i = 1$ to n do

OP_i

Time: $t_1, t_2, t_3, \dots, t_n$

(2) Design of algorithms or data structures

(a) Traditional:

- ✓ Try to reduce $f(n)$ (worst case of each t_i)
- ✓ Every t_i should be small

(b) Amortized:

- ✓ Try to reduce $T(n)$ (overall running time)
 - ✓ Most t_i are small
 - ✓ But, allow a few t_i to be large
 - ✓ Have more flexibility in designing
 - ✓ Have more chance to get a better $T(n)$
- (See CH21: disjoint sets)

