Amortized cost：一種表示法

用"每一個人的平均"去表示 "全部加總"

Example:

A copy machine

OP_1	0.5~1 sec.
OP_2	0.5~1 sec.
⋮	⋮
OP_{499}	0.5~1 sec.
OP_{500}	0.5~1 + 120 sec.
OP_{501}	0.5~1 sec.
OP_{502}	0.5~1 sec.
⋮	⋮
OP_{1000}	0.5~1 + 120 sec.

Single operation

best-case: 0.5
(沒意義，廣告詞)

worst-case: 121 ⇨ $T(n) \le 121 * n$
(有品質保證，但太悲觀)

amortized: 1.24 ⇨ $T(n) \le 1.24 * n$
(這個表示法最好)

Note: amortized ≠ average-case

0.75+120*(1/500)

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

best-case : 1

worst-case : k ⇨ $T(n) \le k * n$

amortized : 2 (最好的表示法) ⇨ $T(n) \le 2 * n$ (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
- ① 1.24 sec. per page (✓)
- ② 620 sec. for 500 pages (×)
- ③ 1.24n sec. for n pages (×)

Aggregate Method

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

Operation	Amortized cost	Actual cost	△credit
X	a_X	t_X	
Y	a_Y	t_Y	
Z	a_Z	t_Z	

① Assign

② How (specific object)

③ prove credit ≥ 0 (for any n)

$\Rightarrow \underset{付}{\sum a_i} \geq \underset{花}{\sum t_i} \quad (\underset{付}{\sum a_i} = \underset{花}{\sum t_i} + \underset{存}{C})$

④ $\underset{花}{T(n)} = O(\underset{付}{\sum a_i})$

⑤ Compute ④/n

Problem: ① and ② are not easy!

Potential Method

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

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

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

OP	Amortized cost	Actual cost	△credit
X	a_X	t_X	
Y	③ compute 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

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

$\sum a_i \geq \sum t_i$ （付 ≥ 花）

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

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

付 花 存 款 變 化

④ $T(n) \leq \sum a_i \leq 7n$

⑤ $T(n)/n \leq 7n/n \leq 7$

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

$\alpha = \frac{1}{2}$ $\Phi = 0$

a | b | c | d

$t_i = 1$ $a_i = 3$ $\Phi = 2$

a | b | c | d | e

$t_i = 1$ $a_i = 3$ $\Phi = 4$

a | b | c | d | e | f

$t_i = 1$ $a_i = 3$ $\Phi = 6$

a | b | c | d | e | f | g

$t_i = 1$ $a_i = 3$ $\Phi = 8$

a | b | c | d | e | f | g | h

$t_i = 9$ $a_i = 3$ $\alpha = \frac{1}{2}$ $\Phi = 2$

a | b | c | d | e | f | g | h | i

每多一個存2塊

Insert $a_i \leq 3$

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

Delete $a_i \leq 2$

$\alpha = \frac{1}{2}$ $\Phi = 0$

a | b | c | d | e | f | g | h

$t_i = 1$ $a_i = 2$ $\Phi = 1$

a | b | c | d | e | f | g

$t_i = 1$ $a_i = 2$ $\Phi = 2$

a | b | c | d | e | f

$t_i = 1$ $a_i = 2$ $\Phi = 3$

a | b | c | d | e

$t_i = 1$ $a_i = 2$ $\Phi = 4$

a | b | c | d

$\alpha = \frac{1}{2}$ $\Phi = 1$ $t_i = 4$ $a_i = 1$

a | b | c

每少一個存1塊

Another viewpoint -- Accounting Method

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

$\alpha = \frac{1}{2}$ $\Phi = 0$

a | b | c | d

\$1 \$1 $\Phi = 2$

a | b | c | d | e

\$1\$1 \$1\$1 $\Phi = 4$

a | b | c | d | e | f

\$1\$1\$1 \$1\$1\$1 $\Phi = 6$

a | b | c | d | e | f | g

\$1\$1\$1\$1 \$1\$1\$1\$1 $\Phi = 8$

a | b | c | d | e | f | g | h

\$1 \$1 $\Phi = 2$

a | b | c | d | e | f | g | h | i

每多一個存2塊

Insert $a_i = 3$

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

Delete $a_i = 2$

$\alpha = \frac{1}{2}$ $\Phi = 0$

a | b | c | d | e | f | g | h

\$1 $\Phi = 1$

a | b | c | d | e | f | g

\$1\$1 $\Phi = 2$

a | b | c | d | e | f

\$1\$1\$1 $\Phi = 3$

a | b | c | d | e

\$1\$1\$1\$1 $\Phi = 4$

a | b | c | d

\$1 $\alpha = \frac{1}{2}$ $\Phi = 1$

a | b | c

每少一個存1塊

Amortized cost : 一種表示法

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)
DS_2	O(lgn)	O(lgn)

good for lib (or a group of users)

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 ⇨ 每次都很好(快)
* single-operation amortized cost 好

⇨ 整體表現好(偶爾很差(慢))

⇨ 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

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

(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: $t_1, t_2, t_3, \ldots, t_n$

(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)